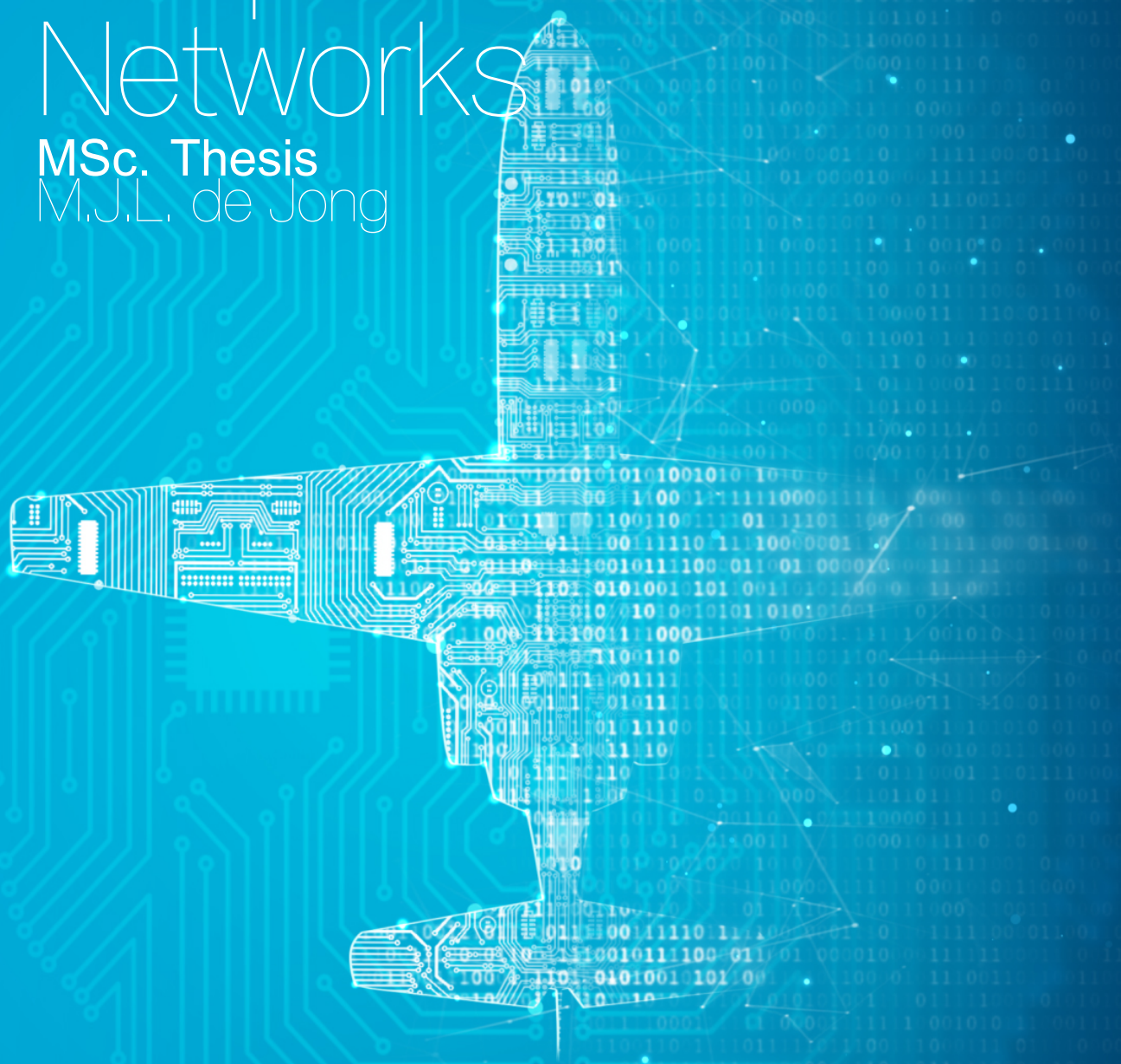
Classifying Human Pilot Skill Level Using Deep Artificial Neural Networks

MSc. Thesis

M.J.L. de Jong

Delft University of Technology



TU Delft

Classifying Human Pilot Skill Level Using Deep Artificial Neural Networks

MSc. Thesis

by

M.J.L. de Jong

to obtain the degree of
Master of Science in Aerospace Engineering
at Delft University of Technology

Student number:	4303229		
Date:	November 29, 2021		
Thesis committee:	Prof. dr. ir. M. Mulder,	Delft University of Technology,	Supervisor
	Dr. ir. D. M. Pool,	Delft University of Technology,	Supervisor
	Dr. O. A. Sharpanskykh,	Delft University of Technology,	External Member
	Dr. ir. M. M. van Paassen,	Delft University of Technology,	Additional Member

TU Delft

Preface

This MSc thesis report details my work over the past year, researching a 'human pilot skill level identification method using explainable deep artificial neural networks', to conclude my Master of Science in Aerospace Engineering at Delft University of Technology. The report consists of three parts: (I) a scientific paper, (II) appendices to the scientific paper, and (III) a preliminary report (consisting of a literature study and a preliminary experiment).

The past year—working on this thesis—has been a challenging, ambitious, but very rewarding experience. Challenging, because of the global pandemic forcing everyone to work isolated from home. Ambitious, because of taking on a research topic (artificial intelligence) out of my comfort zone. And rewarding, because of everything that I have learned over the past year under the guidance of my amazing supervisors Max Mulder and Daan Pool. My sincerest appreciation goes to their 24/7 support (answering my emails any time of the day, any day of the week), their positive and inventive attitude during our meetings (always bringing new insights and ideas to the table), and their incredibly detailed feedback on my work (elevating my research and my scientific writing to a higher level).

Additionally, I would like to thank all my friends and family for putting up with my *artificial neural network* stories at the dinner table. Being able to talk with them about my excitement for this project upheld my motivation and always set new ideas in motion. I'd like to give special thanks to my parents and brother, who always support me and cheer me on. Without their nonstop encouragement, my time in Delft would not have been as successful.

And thank *you*, the reader, I hope you enjoy this work as much as I did producing it.

M.J.L. de Jong
Den Haag, November 2021

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

AI	Artificial Intelligence
AM	Activation Maximization
ANN	Artificial Neural Network
BN	Batch Normalization
CAM	Class Activation Map
CNN	Convolutional Neural Network
DGN	Deep Generative Network
DNN	Deep Neural Network
FCN	Fully Convolutional Network
GAN	Generative Adversarial Network
GAP	Global Average Pooling
GPU	Graphics Processing Unit
IncTim	InceptionTime
LIME	Local Interpretable Modelagnostic Explanations
LSTM	Long Short-Term Memory
ML	Machine Learning
MTS	Multivariate Time Series
NN	Neural Network
PDF	Probability Density Function
ReLU	Rectified Linear Unit
ResNet	Residual Network
RMS	Root Mean Square
RNN	Recurrent Neural Network
SHAP	SHapley Additive exPlanations
TSC	Time Series Classification
UPN	Unsupervised Pretrained Network
UTS	Univariate Time Series
XAI	eXplainable Artificial Intelligence

Symbols

α	learning rate
δ_e	elevator deflection
$\hat{y}$	output prediction
ω_b	break frequency
ω_d	disturbance frequencies
ω_m	measurement time base frequency
ω_{nm}	neuromuscular frequency
ω_t	target frequencies
ϕ	activation function
ϕ	phase
ρ	Pearson's correlation coefficient
σ	sigmoid function
σ^2	variance
τ_m	motion time delay
τ_v	visual time delay
θ	network trainable parameters
θ	pitch angle
$\tilde{C}_t$	LSTM potential cell state
ζ_{nm}	neuromuscular damping ratio
A	amplitude
a	activation value of neuron
b	bias of neuron
C	output of convolutional layer
C_t	LSTM cell state
D	data set
d	depth of artificial neural network
dt	time step size
E	expected value
e	tracking error
f	approximation function
f_t^*	LSTM remember vector

f_d	disturbance function
f_s	sampling frequency
f_t	forcing function
h	hidden state
H_{θ,δ_e}	elevator-to-pitch dynamics
H_{nm}	neuromuscular dynamics
H_{p_m}	pilot motion response
H_{p_v}	pilot visual response
H_p	pilot describing function
H_{scc}	semicircular canal dynamics
i_t	LSTM save vector
K	gain
K	number of classes
K	number of filters
K_m	motion gain
K_s	stick gain
K_v	visual gain
L	kernel size
L	loss
L	number of recorded time steps
L_w	labeling width
M	amount of output neurons
M	number of time series
N	scalar amount of points in the dataset
n	remnant
o_t	LSTM cell potential output
S_f	sampling frequency
s_i	output value of neuron i in dense layer
S_{xx_y}	power-spectral density of x due to y
S_{xx}	power-spectral density of x
SF	sampling frequency
T	total length of time series
T_{lag}	visual lag time constant
T_{lead}	visual lead time constant
T_l	lag time-constant
U	trainable weight in RNN cell
u	pilot control output
V	trainable weight in RNN cell
W	trainable weight in RNN cell
w	weight between neurons
WS	window size
X	time series vector
x	controlled element output
x	input
Y	label vector
y	output
z	output of the hidden neuron

Scientific Paper

Real-time Explainable Human Control Skill Classification Using Deep ANN

M.J.L. de Jong, *Author*, D.M. Pool, *Supervisor*, and M. Mulder, *Supervisor*

***Abstract*—To fully optimize the synergy between human operators and machines in modern day's highly automated vehicle control tasks, a real-time quantitative feedback of skill level is required. Direct feedback of skill level could be used to enable scalable levels of autonomy of the controlled system, or to provide a warning when sudden skill level deterioration is detected, improving safety. Cybernetics has proven to be a useful tool to assess pilot skill level, but most traditional methods suffer from the fundamental issue of assuming the human controller to be constant over time, ignoring subtle changes in control behavior. Employing deep artificial neural networks directly to raw time series of control behavior may be a solution to this problem. Using a deep residual convolutional neural network (ResNet) architecture, this research shows that 1.2 second windows of experimental human control data—from a previously conducted compensatory tracking experiment in the SIMONA Research Simulator at Delft University of Technology—can be classified as either 'skilled' or 'unskilled' with an average validation accuracy of 92% in a moving-base setting and 88% in a fixed-base setting. Results indicate that the trained network is not a one-size-fits-all classifier in its current state, as the skill levels of isolated subjects with off-nominal learning curves are difficult to predict. Introduction of SHapley Additive exPlanations and class visualizations with activation maximization add transparency to the trained classifier's predictions, offering a new perspective on distinctive characteristics of manual control skill level in the time domain. This explainable deep learning approach to skill level identification enables real-time quantitative evaluation of control behavior, opening a new realm of possibilities to enhance safety in automated systems that rely on smooth interaction with the human operator.**

Keywords: Manual control, time series classification, deep learning, explainable artificial intelligence, pilot modeling, skill level

I. Introduction

Automation in manual control has become a common element in almost all human-machine interactions, from autonomously driving cars, to today's highly automated aircraft. Although automation can enhance control performance and reduce workload, it can also reduce situation awareness of the human operator and degrade the ability to regain manual control. Across all domains, there is a raised concern about the integrity of manual control of the human operator in these highly automated systems [1–4]. Although some research conclude that the manual control skill of human operators remains—for the most part—intact, there are also aviation industry experts and pilots who are concerned with the increased reliance on automation and question whether pilots are provided enough training and experience to maintain manual flying proficiency [5, 6]. A real-time quantitative measure of manual control skill level would greatly benefit the evaluation of pilot skill retention and improve safety. Direct feedback of skill level could also be used to improve the human-machine synergy through scalable levels of autonomy, or to provide a warning signal whenever skill level deterioration is detected.

Effectively quantifying a pilot's skill level in a manual control task is no easy task. Cybernetics has proven to be a useful tool to assess pilot skill level [7, 8], but most of its currently available methods suffer from the fundamental limiting requirement of assuming the human controller to be constant over a substantial time basis, ignoring any subtle changes in control behavior [9]. Although efforts have been made to model the time-varying nature of the human controller [10–13], capturing short-duration temporal variations in the pilot control behavior from inherently noisy data remains a difficult task [14]. Recent studies [15, 16] have investigated the applicability of artificial neural networks to solve this difficult task. Their work concluded that exploiting deep learning models in time series classification [17] may be a solution to recognize human adaptive control behavior in real-time.

Although recent years have shown a large amount of research on machine learning applications for identifying behavior of the human controller [18–21], the identification of pilot skill level using artificial intelligence remains relatively untouched. Xi et al. [22] employed machine learning to identify pilot skill level, but their work required manual input feature engineering and the resulting model was most accurate when using 60 s of data, falling into the time range where classic cybernetic methods would be equally effective [14]. Additionally, all of the aforementioned research requires the use of additional sensors such as face cameras, eye tracking devices, outside cameras, and electrocardiograms, making the implementation of these methods impractical. Utilization of the automatic feature extraction capability of deep networks on raw time series control data may be the key to real-time pilot skill level identification without the need of additional apparatuses. Deep learning may be especially useful for the classification of *untrained* pilots, since they are known to exert inconsistent nonlinear control behavior [23], also known as *remnant*. Classic cybernetic evaluation usually ignores this portion of the manual control behavior [8], but the automatic feature extraction capabilities of deep learning may incorporate it in its skill level prediction, utilizing the full range of pilot control behavior.

This paper will introduce a novel *pilot skill level classification* method using deep learning with only readily available task related time signals (e.g., pilot control input, tracking command, and tracking error). A deep artificial neural network

is trained to classify raw time series data of either 'skilled' or 'unskilled' pilots. Due to the few (and rudimentary) time-domain input signals required, this method would not need any additional sensors to be added to the cockpit of most modern aircraft. Experimental human control data from a previously conducted compensatory tracking experiment by Pool et al. [8] is used to train and validate the classifier in this paper. These data contain time traces of human pilots performing training in a pitch tracking task—with and without motion feedback—in the SIMONA Research Simulator at Delft University of Technology. To expand the body of knowledge on the time-varying nature of the human controller, the skill level predictions of the trained classifier will also be interpreted using eXplainable Artificial Intelligence (XAI) techniques [24]. SHapley Additive exPlanations (SHAP) [25] are used to compute the input feature importance of local predictions, which can be aggregated into global explanations. Activation Maximization (AM) [26] is utilized to visualize the internal class representations of the trained classifier, so that class specific input patterns can be recognized.

The paper is structured as follows. Section II introduces the data sets used and data handling steps taken. Section III explains the optimization of the proposed deep artificial neural network classification method. Section IV details the explainability methods utilized to make the deep learning model interpretable. The results of this research are presented in Section V. All results are discussed in Section VI. Section VII concludes the paper.

II. Data Sets and Data Handling

A. Data Set

This paper studies the applicability of deep learning to distinguish raw time signals of untrained versus trained pilot behavior. As such, a data set is required that contains pilot control behavior time traces of both inexperienced and experienced individuals performing the same task.

In this paper, data from an experiment by Pool et al. [8] are used. This experiment, which was performed in the SIMONA Research Simulator at Delft University of Technology, studied the effects of simulator motion feedback on the training of multimodal skill-based human operator control behavior. Here, 24 fully task-naive participants underwent training in a compensatory pitch tracking task designed for multimodal human control behavior identification. In the experiment, consisting of a training- and evaluation-phase, the participants were split into two groups. The first group was trained in a *fixed-base* setting with only a visual indication of their tracking error e, shown on a compensatory display (example shown in Figure 1), before being transferred to a *moving-base* setting for evaluation. The second group was trained in a *moving-base* setting (in addition to the compensatory display), before being evaluated on a *fixed-base* setting. Each participant completed a total of 100 training runs and 75 evaluation runs, with each run lasting 90 s of which the last 81.92 s were used as measurement data.

A schematic depiction of the pitch tracking task is shown in Figure 1. Here e indicates the tracking error, u the human control input, and x the controlled state (i.e., the pitch angle). The dotted line connecting x directly to the human pilot is only active if motion feedback is provided. As illustrated in Figure 1, this tracking task requires the human operator to follow a target pitch angle as accurately as possible, while rejecting an additional disturbance signal placed on the controlled element. The controlled element in this experiment simulated the elevator-to-pitch dynamics of a Cessna Citation I, by means of the reduced-order linearized model given in Eq. (1) [27]. This combined target-following and disturbance-rejection task enables reliable separation and identification of the human operator response to visual cues and motion cues [28]. The described task design was substantiated by a number of previous studies that successfully identified multimodal human control behavior [27, 29, 30]. An example of the time traces recorded during one tracking run are shown in Figure 2.

$$H_{\theta,\delta_e}(s) = 10.62\frac{s+0.99}{s\left(s^2+2.58s+7.61\right)} \tag{1}$$

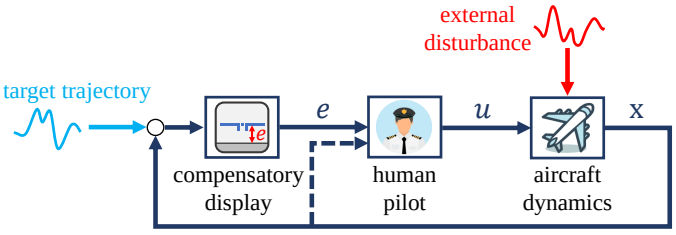


Fig. 1. Schematic depiction of human pilot performing pitch tracking task.

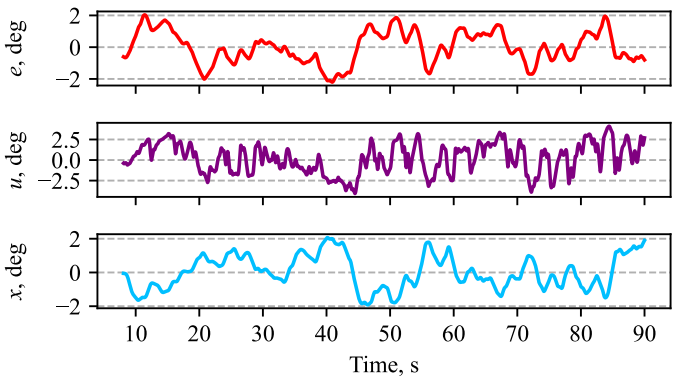


Fig. 2. Example time traces of one tracking run.

Evidently, the data acquired during the training portion of the experiment by Pool et al. [8] are a fitting selection for the objective of this machine learning pilot skill level classification research, as it contains time traces of both *inexperienced* human control behavior (at the start of training) and *experienced* control behavior (at the end of training). Both the fixed-base and moving-base data are used, these data sets are referred to as Data-NM (No Motion) and Data-M (Motion), respectively. Each of these data sets contains a total of 98,304 s of pilot tracking data recorded at 100 Hz.

B. Data Preparation and Preprocessing

Although this research aims to utilize the capability of deep learning models to handle raw time series data (i.e, without the need of manual feature engineering), a few simple data preparation and preprocessing steps have to be taken to optimize model performance and attain desired classification behavior. First, the input time series must be scaled to improve convergence [31]. Specifically, standardization is used to scale the input data, as this method was found to be the most effective in a similar classification study [15]. Second, a number of data handling steps with adjustable settings are taken.

To understand the adjustable data handling steps taken in this research, first observe the classification process as summarized in Figure 3. As illustrated, the input to the classifier is an observed window of the previously introduced e, u, and x time signals. Every observed window (or *sample*) has two dimensions: the number of time steps and the number of input variables. The outputs of the classifier, $P(Y_1)$ and $P(Y_2)$, indicate the classifier's predicted probabilities that the observed window belongs to class Y_1 or class Y_2 (i.e., 'skilled' or 'unskilled' in this paper). The presented task definition opens some questions, namely: 1) What is the definition of 'skilled'/'unskilled' behavior and how are the data *labeled* accordingly? 2) What is the best *window size* and at what *sampling frequency* is it observed? And 3) which combination of *input variables* should be used? The following sections will elaborate on the relevance of these questions and explain the methods that are used to answer them in this paper.

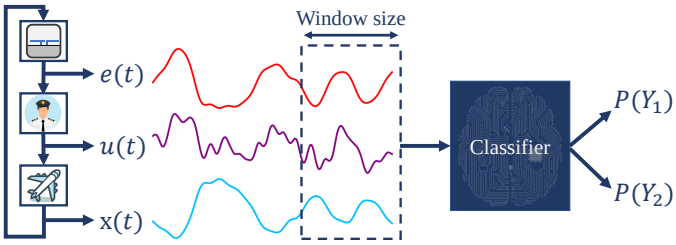
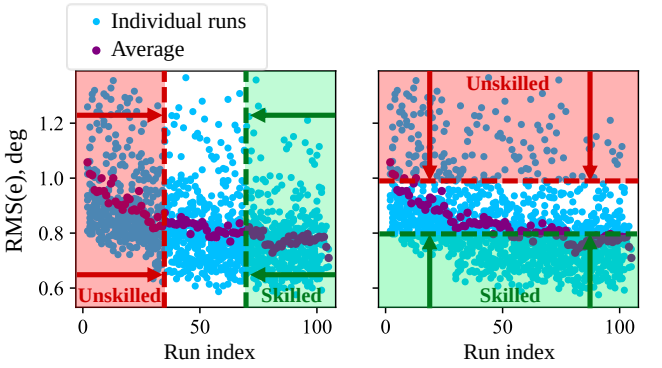


Fig. 3. Schematic description of pilot skill level classification task.

1) Labeling the Data: As this is a supervised learning task, every sample of pilot control data requires a corresponding class label to enable training of the classifier. Two distinct labels (and therefore classifier outputs) are considered in this research: 'skilled' and 'unskilled' pilot control behavior. This raises the question of what the definition of skill level is and how it is identified. Two options to define skill level (and thereby label the data) were considered: 1) every run is labeled based on the level of experience (by the run index, Figure 4a) or 2) every run is labeled based on a performance measure (e.g., the root mean square tracking error, Figure 4b). Preliminary research [32] concluded that using experience-based labeling resulted in the desired classification behavior.

The amount of runs that are labeled as 'unskilled' and 'skilled' (i.e., the lengths of the arrows in Figure 4a) is determined empirically. In this paper, this parameter is referred to as the labeling width L_w. The labeling of data is done symmetrically (equal amount of samples under each class), to prevent class imbalance and potential bias in the trained classifier [33, 34]. After the data are labeled, they are randomly split into a train (80%) and validation (20%) set. The validation accuracy of the trained classifier is used to compare—and ultimately optimize—different hyperparameter settings.

A drawback of the experience-based labeling method is that it provides a single class label per subject tracking run. Ideally, the data would be labeled per *sample*, since the human pilot is a time-varying system that may exert variations of control character due to factors such as fatigue, loss-of-attention, and learning [35, 36]. Seeing these factors are considered to induce "slow" variations in control behavior [37] and because they are not known a priori when the data are labeled, the human pilot is assumed to be time-invariant per tracking run for labeling. This assumption has to be made, as alternative methods to determine these changes in control behavior are currently lacking or nonexistent [14]. Considering there were no sudden changes to the controlled environment in the experiment by Pool et al. [8], assuming the human controller to be time-invariant per tracking run is reasonable [14].



(a) Experience-based labeling method. (b) Performance-based labeling method.

Fig. 4. Visual explanation of the two considered labeling methods.

2) Window Size and Sampling Frequency: As was shown in Figure 3, the classification of time series is done per observed window of a specified size. This technique, also known as *window slicing* [38], takes a time series T with N time steps $T = \{t_1, \ldots, t_N\}$ and produces multiple (shorter) snippets defined as $S_{i:j} = \{t_i, t_{i+1}, \ldots, t_j\}$, $1 \leq i \leq j \leq N$. The size of each slice is a parameter of this method that is empirically optimized in the Results section.

Window slicing is beneficial to the goal of this research for two reasons: 1) Training the classifier model with short time series samples will result in a trained classifier that can predict the class of short recordings of pilot control behavior. This allows for real-time feedback, enabling applications such as sliding scale autonomy or skill deterioration warnings. 2) By extracting a larger set of smaller snippets from the original time series, more training data become available to the classifier. Increasing the amount of training data, also known as data augmentation, helps to avoid overfitting and improve the

generalization ability [38, 39]. This data augmentation aspect of window slicing can be further exhausted by overlapping consecutively sliced windows, as is shown in Figure 5.

The amount of time steps in a fixed window size is determined by the *sampling frequency*. Originally, the experiment data from Pool et al. [8] were recorded at 100 Hz. Downsampling these data, especially at larger window sizes, may be advantageous for good classification performance [15]. In the Results section of this paper, it is shown how well the classifier performs under different combinations of window sizes and sampling frequencies.

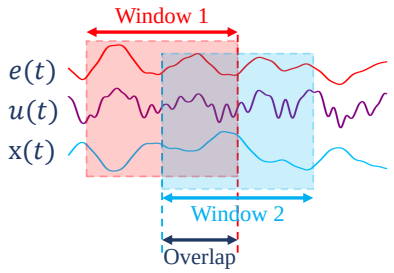


Fig. 5. Consecutively sliced windows can be overlapped to produce more training samples.

3) Input Variables: The last data preparation setting to be researched, is the selection of input variables. As previously explained, there are three input variables in the original experiment data set of Pool et al. [8]: e, u, and x. However, following the suggestion by Versteeg [15], three additional input variables can be introduced as the first-order time derivatives of these signals, i.e., $\dot{e}$, $\dot{u}$, and $\dot{x}$. These time derivatives are calculated using a central difference approximation (and one-sided forward or backward approximation at the boundary points of each tracking run). This provides a total of *six* potential input variables, which results in *63* possible unique input combinations. Each of these combinations is tested in the Results section.

To conclude this section on data preparation and preprocessing, the initial *data* hyperparameter settings (before optimization) are provided in Table I. These settings were either taken from Versteeg [15] or found during the preliminary phase of this research [32]. Additionally, the values that are tested during optimization are also indicated. The overlap will not be optimized, because it was discovered during preliminary testing that higher overlap always resulted in better classification performance [32]. The effect of different train/validation splits is not investigated.

TABLE I
DATA HYPERPARAMETERS.

Setting	Symbol	Baseline value	Values to be tested
Train/validation	-	80% / 20% [15]	*fixed*
Overlap	-	90 % [32]	*fixed*
Labeling width	L_w	15 runs [32]	[1 run - 40 runs]
Window size	WS	1.6 s [15]	[0.1 s - 30 s]
Sampling frequency	S_f	50 Hz [15]	[25 Hz - 100 Hz]
Input variables	-	e+$\dot{e}$+u+$\dot{u}$ [32]	[e, $\dot{e}$, u, $\dot{u}$, x, $\dot{x}$]

III. Time Series Classification using Deep Artificial Neural Networks

A. Deep Neural Networks

Traditionally, Time Series Classification (TSC) has been carried out using conventional machine learning techniques [40]. However, these conventional machine learning techniques often require thorough engineering—and substantial domain expertise—to design feature extractors that transform the raw data into internal representations (or feature vectors) from which a classifier could detect patterns in the input [41]. Contrarily, *deep learning* methods automatically discover the representations needed for classification by composing modules that each transform the representation into a higher, more abstract, level, starting with the raw input and ending at a class representation [41]. In relatively recent work [17, 42], performance comparisons have been made between traditional TSC methods and deep learning TSC methods. From this work it was found that deep learning methods can achieve comparable—and sometimes better—performance than state-of-the-art traditional TSC methods (with the added benefit of not requiring manual feature engineering). Based on these findings, this research will limit its scope to deep learning methods only.

1) Architectures: The implementation of the deep network architectures is done in *Python v3.8*, using the artificial neural network library *Keras v2.6.0* [43] running on top of *TensorFlow v2.6.0* [44]. The following four deep learning architectures are tested as candidates:

a) Long Short-Term Memory (LSTM): These are a special kind of Recurrent Neural Network (RNNs) introduced by Hochreiter and Schmidhuber [45] and specifically built to overcome the problem of "vanishing gradient" [46], allowing the learning of long term context. In the work of Versteeg [15], a stacked LSTM model was used to perform TSC on pilot control data. This stacked LSTM structure was inspired by the architecture used by Saleh et al. [19], combined with the rule of thumb hyperparameter settings from Reimers and Gurevych [47]. By stacking LSTM layers the network becomes deeper. This way the learned representation from the first layer can be passed on to the next layer, which creates representations at a higher level of abstraction. Each layer will take on a part of the task and pass it to the next, until finally the last layer provides the output [48].

The model that is tested here, consists of two stacked LSTM layers, each containing 100 LSTM cells. Both layers use dropout [49] as regularization method to decrease overfitting. Lastly, the output of the second (last) LSTM layer is fully connected to two output neurons with softmax activation. These settings are optimized for sequence labeling tasks [47] and have been successfully used for the classification of human manual control time traces [15, 32].

b) Fully Convolutional Networks (FCN): Long et al. [50] showed that end-to-end trained FCNs—without further

machinery—exceeded state-of-the-art performance in semantic mapping. Wang et al. [42] adopted the FCN architecture as a feature extractor for TSC, with its final output coming from a softmax layer. Simply stacking convolutional layers, without pooling operations in between, turned out to be an effective architecture for TSC [17, 42].

The aforementioned TSC FCN architecture consisted of three *convolutional blocks*. Here each convolutional block is made out of a convolutional layer with K_n filters with kernel size L_n, followed by a Batch Normalization (BN) layer [51] and a ReLU activation layer [52]. Specifically, the number of filters between the three blocks are $\{128, 256, 128\}$, with 1D kernels with sizes $\{8, 5, 3\}$. The three blocks are followed by a Global Average Pooling layer (GAP) [53] instead of a fully connected layer, to largely reduce the amount of weights [42]. Lastly, the final class label is produced by a softmax layer.

c) Residual Networks (ResNet): He et al. [54] introduced this class of networks as a framework to train deep neural networks for image recognition. Simonyan and Zisserman [55] showed that adding more convolutional layers (thus making the network deeper) is beneficial for the accuracy in image classification. However, adding more layers also makes training of the network more difficult. He et al. [54] developed a method that uses 'shortcut' residual connections to overcome these training difficulties, whilst still profiting the enhanced accuracy from increased network depth.

Wang et al. [42] adopted ResNet for TSC, the resulting structure (as displayed in Figure 7) is somewhat similar to the previously described FCN architecture. However, now each block is replaced by a *residual block* containing three convolutional layers with an equal amount of filters per block, but with varying kernel sizes, again followed by a BN and ReLU layer. A shortcut connection combines the input of the first convolutional layer with the output of the third (last) convolutional layer, before passing it through the final BN + ReLU layer. Taking the proposed settings from Wang et al. [42]; the three residual blocks have $\{64, 128, 128\}$ number of filters, with 1D kernel (filter) sizes $\{8, 5, 3\}$ per block.

d) InceptionTime (IncTim): Out of the network architectures to be tested, this is the most recent development. Fawaz et al. [56] introduced InceptionTime as a novel architecture specifically designed for TSC. This network is similar to the TSC adaptation of ResNet, in the sense that it also uses residual blocks and a GAP layer followed by a fully connected layer with softmax activation. However, there are *two* residual blocks, instead of *three*, and each of the residual blocks is composed of three Inception modules [57] instead of the traditional convolutional layers. What is unique about the Inception module is that it uses multiple sliding filters with an increasingly large receptive field that simultaneously extract features from the same time series. For a more elaborate explanation of this architecture, the reader is referred to the original work [56]. This network is tested here with the default settings as they were proposed by Fawaz et al. [56].

2) Optimization: Two levels of optimization are discussed: trainable parameter optimization and hyperparameter optimization. Trainable parameters are internal to the model, they are learned from data during training as the model tries to map input to output. Hyperparameters are parameters that control the learning process, these can be tuned to make networks train better and faster [58].

a) Trainable parameter optimization: In order to optimize the internal model parameters, there must first be a definition of loss. As was depicted in Figure 3, the output of the classifier is a probability distribution over the possible classes. Attaining such a probability distribution is achieved by using a *softmax* activation function over the two output neurons of the classifier. To optimize for maximum likelihood (i.e., predicting the most likely probability of each class) a categorical cross-entropy loss function is used [59]. The cross-entropy loss L for N samples with M classes can be calculated as follows:

$$L(y, \hat{y}) = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot \log \hat{y}_{ij}, \tag{2}$$

where y is the ground truth and $\hat{y}$ is the model prediction.

Trainable parameter optimization is done by minimizing the loss, using *Adam* optimiser [60] with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-7) and learning rate $\alpha =$ 5e-4 (finding the optimal learning rate is discussed in Appendix B-2). The updated trainable parameter values will only be saved if the *validation* loss improved with respect to the previous epoch.

b) Hyperparameter optimization: There are two categories of hyperparameters that are optimized: those that pertain to the data handling (Section II-B, Table I) and those that pertain to the neural network architecture and training (Section III-B, Table II). The method of optimization is the same for both these categories. Namely, different combinations of hyperparameters are tested to empirically determine which settings result in the best model classification performance.

The validation accuracy is used as the measure of model performance. The models are trained for 20 epochs with a batch size of 128, here 80% of the data are used for training and 20% are used for validation. Each model is evaluated at the epoch with minimal validation loss. To account for the stochastic training outcome, every network is tested 30 times and the average performance is used as final comparison measure. For fair evaluation, every network is tested under the same 30 random train/validation splits.

3) Selection: The first step in the optimization process is selection of the classifier. Therefore an empirical performance comparison between the different classifiers—with their default hyperparameters (as described in Section III-A1)—is made, leaving only one model to be considered for model-specific hyperparameter optimization. The models are compared using the performance comparison method described in Section III-A2 with the data settings kept at their initial values (Table I). The results of this performance comparison are shown in Figure 6. Based on these results, ResNet is selected as the appropriate architecture for time series classification of pilot skill level.

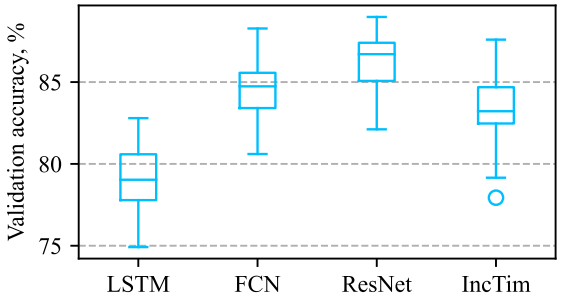


Fig. 6. Performance of different deep learning architectures. N = 30, *Data set = Data-NM*

B. ResNet Hyperparameters

The following model-specific hyperparameters are optimized: depth of network d, number of filters K, and filter (kernel) length L. In order to explain these hyperparameters, the fundamental working principal of a 1D convolutional layer is described first.

Figure 8 displays how a 1D convolutional layer with K filters of length L sliding across an input X with step size (stride) S, produces an output C. If X is Multivariate Time Signal (MTS) of length N_{in} with c channels, then a 1D convolutional layer with K filters outputs C: an MTS with K feature maps of length N_{out}, where every element in $C_{t,m}$ at time step t on feature map m is given by Eq. (3). Here k_m is a kernel of size L by c, where L is the user-defined kernel length and c is equal to the number of channels of the input signal X. Every kernel has $c \times L$ trainable weights $w_{m\{1:c,1:L\}}$ and a bias b_m.

$$C_{t,m} = X_{t-L/2:t+L/2} \cdot k_m + b_m \quad \text{for } t = [1, N_{out}] \text{ and } m = [1, K] \tag{3}$$

A TSC-adapted-ResNet [42] architecture is shown in Figure 7. As was explained in Section III-A1, this architecture consists of multiple residual blocks. The number of residual blocks are referred to as the depth d of the network. Each residual block has three 1D convolutional layers with varying kernel lengths L_1, L_2, L_3, but an equal number of filters K. The length of the output N_{out} of each convolutional layer is a function of input length N_{in}, kernel size L, and stride S. In this implementation, S is set to 1 for all convolutional layers, and padding is used to ensure equal length of the input and output ($N_{in} = N_{out}$).

The default hyperparameter settings, as proposed by Wang et al. [42], are as follows: network depth $d = 3$, with number of filters $K = \{64, 128, 128\}$, and kernel sizes $L = \{8, 5, 3\}$. These hyperparameters are optimized by using a grid search. Networks up to a depth of 4 are tested, where the number of filters double at every even residual block. For example, if $d = 4$ and $K_1 = 64$, then $K = \{64, 128, 128, 256\}$. The kernel sizes to be tested are $\{5, 3, 2\}$, $\{8, 5, 3\}$, $\{16, 10, 6\}$, and $\{32, 20, 12\}$. All hyperparameter values are summarized in Table II.

TABLE II
MODEL HYPERPARAMETERS.

Setting	Symbol	Baseline value	Values to be tested
Number of epochs	-	20	*fixed*
Batch size	-	128	*fixed*
Learning rate	α	5e-4	*fixed*
Depth	d	3	[1, 2, 3, 4]
First number of filters	K_1	64	[16, 32, 64]
First kernel size	L_1	8	[5, 8, 16, 32]

C. Performance Analysis

Once all hyperparameters are optimized, an evaluation of the performance of the trained classifier is performed. To get a fair measure of performance, a third data split is introduced: the *test* split. This means there will now be a *training* set to fit the model by learning the trainable model parameters, a *validation* set to keep track of validation loss during training to prevent overfitting, and a *test* set to evaluate the classification accuracy of the trained model (i.e., *test* accuracy). Using the *validation* accuracy as a performance measure may yield inflated results because—although the model has never 'seen' the validation data during training—its trainable parameters were only saved at minimal validation loss (thus the model has been optimized to perform well on the validation set). The introduction of a test set overcomes this problem.

To construct the test set, control data of one subject is designated as the test data, after which the data of the remaining eleven subjects are again randomly distributed as 80% training data and 20% validation data (i.e., the model used to evaluate Subject 1 was trained on 80%/20% train/validation split of Subjects 2-12, etc.). This process is repeated twelve times, so that the data of every subject have once served the role of test set. Excluding the test subject entirely from the training/validation data provides a realistic representation of how the classifier would perform in a real life application where it is utilized to predict the skill level of pilots it has not 'seen' before. The average test accuracy across all subjects is used as a final measure of classification performance.

IV. Explainable Artificial Intelligence

A. Explanation Methods

In machine learning there is a trade-off between model interpretability and performance [24]. Simpler models (e.g., rule-based learning, linear regression, or decision trees) are

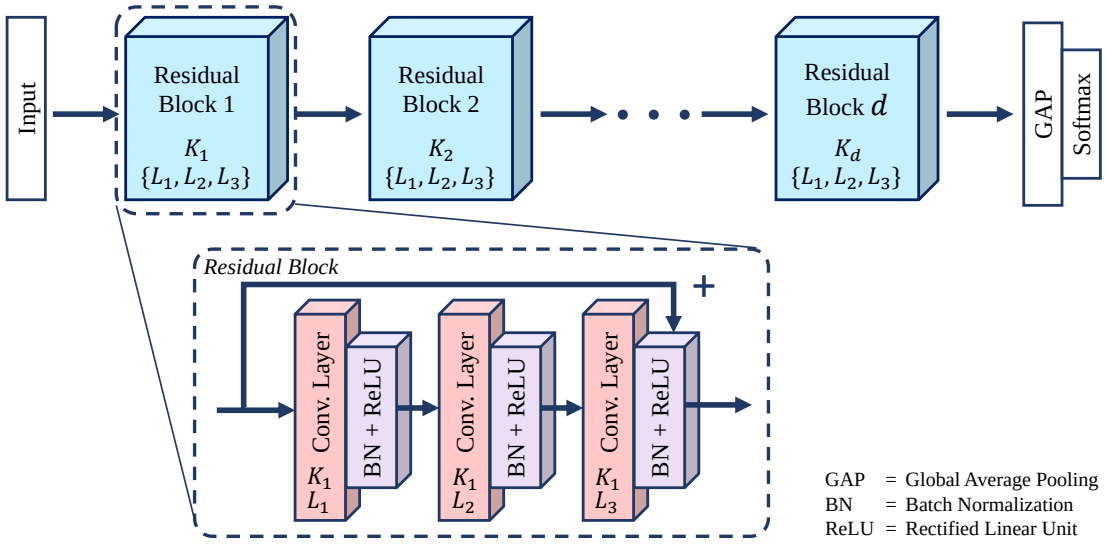


Fig. 7. Network structure of the Residual Network.

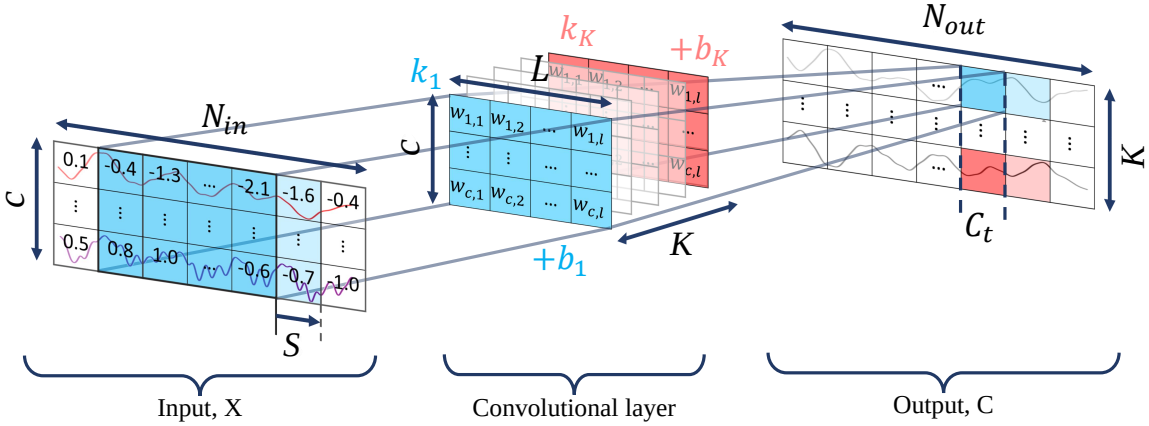


Fig. 8. Forward pass of 1D convolutional layer.

inherently interpretable, but may lack in accuracy [24, 61]. On the other hand, models such as artificial neural networks, support vector machines, and random forests may have better generalization capabilities [61], but their complexity prevents tracing of the logic behind predictions [62]. This means that the explanation of the predictions made by the deep learning classifier in this paper require *post-hoc explainability* [24].

Both model-specific and model-agnostic post-hoc explainability methods were considered to compute the input feature relevance of local skill level predictions. Class Activation Mapping (CAM) [63] and Gradient-weighted Class Activation Mapping (Grad-CAM) [64] were tested as model-specific explanation methods. However, since this paper uses a classifier with 1D convolutional layers, only the temporal dynamics of multivariate time series input are preserved in the forward pass, eliminating the possibility of feature importance computation (this problem is displayed in Appendix D-1). A solution to this problem is using the model-agnostic feature relevance explanation method proposed by Lundberg and Lee [25] called SHAP (SHapley Additive exPlanations). Lundberg and Lee build forth on a game theory approach to compute explanations of model predictions [65–67]. Their work provides feature importance allocations that are more consistent with human inituition than other methods like LIME [68] and DeepLIFT [69].

As a means to interpret the trained classifier's internal representation of 'skilled' and 'unskilled' behavior, the following Convolutional Neural Network (CNN) visualization methods were considered: Activation Maximization [70], Deconvolutional Neural Networks (DeconNet) [71], and Network Inversion [72], each of which has a different purpose. Activation Maximization can be used to produce synthetic inputs that maximize the activation of a selected *neuron*, DeconNet can highlight patterns in an original input that maximize a specific *neuron* activation, and Network Inversion can visualize what input information is preserved in each *layer* of the CNN. Since the latter two methods—like SHAP—provide local explanations, the decision was made to utilize the synthetic class representation generation with activation maximization as CNN visualization method.

B. SHAP

Given an input x and its corresponding model output $\hat{y} = f(x)$, SHAP (SHapley Additive exPlanations) [25] can be utilized to determine the relative contribution of every element in x leading to $\hat{y}$. This approach uses classical Shapley values [73] from cooperative game theory. Given a situation where multiple players contribute to an outcome, cooperative game theory studies how groups of players ("coalitions") can be formed to maximize payoff. In essence, Shapley values are the average expected marginal contributions of each player in all possibly formed coalition. SHAP employs this concept to interpret machine learning model predictions.

SHAP is part of the class of additive feature attribution methods. These methods use an *explanation model*, as shown in Eq. (4), that is a linear function of binary variables each with their own attributed effect ϕ_i [25]. Here g is the explanation model of an original prediction model f, x' are simplified binary inputs that map to the original original input through a mapping function $x = h_x(x')$, and M is the number of simplified input features. The explanation model g is constructed such that it is locally accurate $g(x') \approx f(x)$, i.e., summing the effects of all feature attributions ϕ approximates the output $f(x)$. The base rate $\phi_0 = f(h_x(\mathbf{0}))$ represents the model output with all simplified inputs omitted [25], this will also be referred to as the expected model output $E[f(x)]$.

$$g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{4}$$

For the current pilot skill level classification study, the input X consists of c input variables with N time steps, resulting in $M = N \times c$ input features. To express the relative importance of each input variable i, the marginal contributions at all time steps t are summed first. This results in the expression shown in Eq. (5), where Φ_i indicates the total marginal contribution of input variable i across its sequence length N. The (local) importance of each input variable for a certain prediction can now be expressed as the absolute percentage-wise contribution to the final model output. To illustrate this, consider Figure 9, where it can be seen how every input variable—ranked from the most important (top) to the least important (bottom)—marginally contributes to reach the final model output. The model output, in this case, is of the neuron indicating 'skilled' behavior, i.e., an output of 0.0 means 'unskilled' and an output of 1.0 means 'skilled'. Starting at the expected value $E[f(X)]$, each input sequence X_i 'pushes' the model by a distance $\sum_{t=1}^{L} \phi_{i,t} x'_{i,t}$, resulting in model output $f(X)$. The local % contribution of each input variable (indicated by SHAP:..% in Figure 9) is defined as the absolute distance pushed $|\sum_{t=1}^{L} \phi_{i,t} x'_{i,t}|$ divided by the total distance traveled $\sum_{i=1}^{c} |\sum_{t=1}^{L} \phi_{i,t} x'_{i,t}|$. For example, the local explanation in Figure 9 shows that $\dot{e}$ was the largest contributor to the model output $f(X) = 0.1$. By 'pushing' the model output -0.91, it decided the final prediction by 49%. By performing this analysis for a large quantity of samples, an estimate of global feature importance can be computed.

$$f(X) \approx g(x') = \phi_0 + \sum_{i=1}^{c} \sum_{t=1}^{N} \phi_{i,t} x'_{i,t} \tag{5}$$

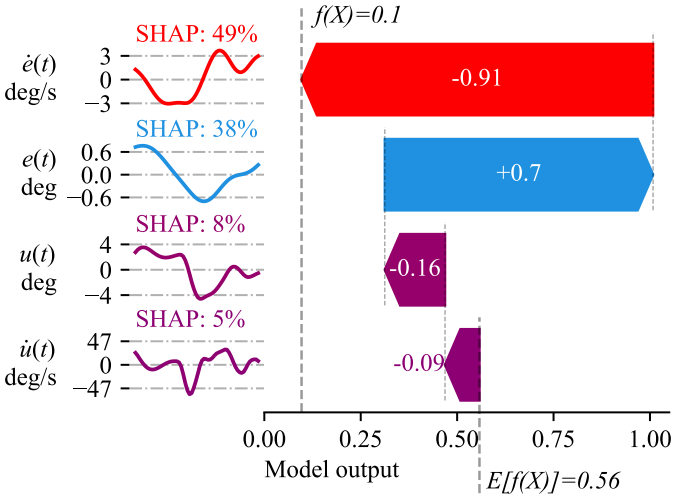


Fig. 9. Waterfall chart displaying the individual contribution of each input channel in X to reach the model output $f(X)$. This a *local* explanation.

C. Activation Maximization

This second XAI method works in the exact opposite direction as SHAP: instead of starting at the input and explaining how it reaches the output, Activation Maximization (AM) starts at the output and computes a corresponding input. This method—as proposed by Erhan et al. [70] to visualize the higher-layer features of computer vision deep networks—looks for input patterns that maximize the activation of a specific neuron. The fundamental AM algorithm is formalized as:

$$x^* = \underset{x}{\operatorname{argmax}}\ a_{i,l}(\theta, x), \tag{6}$$

where $a_{i,l}(\theta, x)$ is the activation of a target neuron i on layer l, for some input image x with a set trainable parameters (weights and biases) θ. For a trained network with fixed θ, the input x is the only parameter set that is updated during the optimization problem, yielding a synthesized input image x^* that maximizes the activation of the target neuron. This process starts with a random input image x_0, after which all pixels in x are updated—through iterations—in the direction that maximizes $a_{i,l}$, i.e., in the direction of the gradient $\frac{\delta a_{i,l}}{\delta x}$.

Simonyan et al. [26] utilized this method to maximize the class scores in the last layer of CNNs. This provides a visualization of the trained model's interpretation of each class. Additionally, Simonyan et al. [26] added L_2 regularization to the optimization function to prevent a small number of extreme pixel values from dominating the class activation, making the image patterns easier to interpret by humans [74]. Here, this implementation is used to visualize features that represent unskilled/skilled control behavior, gaining insight in the model's decision making. A formal expression of the

maximization of score S for class c with L_2 regularization is now given by:

$$X^* = \underset{X}{\operatorname{argmax}}\ S_c(\theta, X) - \lambda \|X\|_2^2, \tag{7}$$

where λ is the regularization parameter. Here X^* is the final synthetic control behavior sequence representing class c.

Lastly, it is important that the class scores S_c to be optimized are unnormalized (i.e., *before* softmax activation). This is because the class scores *after* softmax activation—providing a probability distribution—can be maximized by minimizing the scores of other classes. Optimizing non-normalized S_c ensures that the optimization concentrates only on the desired class c [26]. Effective utilization of this class visualization method will further clarify the trained classifier's internal representation of each 'skilled' and 'unskilled' pilot behavior.

V. Results

A. Optimization

The Data-NM set is used during optimization, so that the effectiveness of this deep learning skill level identification method can be tested in the simplified case where the human controller is of simple, single unity-feedback form [75]. As has been explained in Section III-A2, classification performance with different hyperparameters are tested 30 times to empirically determine the optimal setting. The average performance of the classifier with adjusted hyperparameters is compared against the nominal performance with the baseline hyperparameter settings that were presented in Table I and Table II. Under these nominal conditions, the average validation accuracy of the classifier is **86.35%** on the Data-NM set.

1) Data Settings: Four different data settings are optimized in parallel: the labeling width L_w, the window size WS, the sampling frequency S_f, and the input variable selection.

Labeling width: The labeling width determines the number of tracking runs of each subject that are labeled as either skilled or unskilled, see Figure 4. A very low labeling width means that there are few training samples, but the samples are good representations of their respective class (e.g., a pilot's first run is most likely to show the most unskilled behavior, and vice-versa). A very high labeling width, on the other hand, provides a larger amount of training samples that are, however, less consistent representations of their respective class (pilot control behavior could be either skilled or unskilled around their 50th tracking run). An optimal labeling width will lie somewhere in between these extreme cases.

Eight different labeling widths are tested with all other hyperparameters kept constant. The result of this analysis are shown in Figure 10. Note that $L_w = 1$ is shown on a separate y axis to preserve detail for the other settings. As hypothesized, the extremely low- and high labeling width both have poor average classification accuracy. However, the former has a very large spread in accuracy, whereas the latter has the least spread. This is expected since the high labeling width has a larger quantity of training (and test) samples, and is thus less sensitive to poor class representations in the data (an extended analysis of phenomenon is provided in Appendix B-1). From this optimization analysis it was found that the nominal classification performance can be enhanced by increasing the labeling width from 15 to 20.

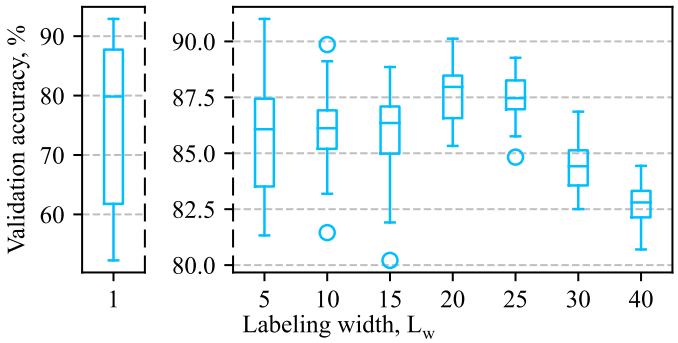


Fig. 10. Classification performance for different labeling widths. N = 30
Data set = Data-NM

Window size and sampling frequency: These two settings are tested concurrently since they conjointly determine the classification performance, as the input length N (number of time steps fed to the classifier) is the product of WS and S_f. Fourteen different window sizes are tested in combination with three sampling frequencies, adding up to a total of 42 different settings (with all other hyperparameters kept at their initial value as presented in Table I and Table II). The resulting validation accuracy of each setting is shown in Figure 11. Two effects can be observed here: 1) for the same overlap, a large window size reduces the amount of training/validation samples that can be drawn from the data set, resulting in more uncertain classification performance (this is especially visible at window sizes 30 s and 40 s), and 2) a high sampling frequency is preferred at small window sizes, whereas a low sampling frequency is preferred at large window sizes. In terms of performance variation across the different WS and S_f, it appears there is little room for improvement. Although a window size of 20 s with sampling frequency of 25 Hz gives the best performance, its usefulness can be argued because this sample size comes into the range of classic cybernetic pilot evaluation [14]. Additionally, to enable direct feedback, a short window size is desired. In Figure 11, it can be seen that performance in the 1-4 second range is somewhat constant. Based on this observation—and with the goal of direct feedback in mind—it was decided to select the shortest window size before performance drops, i.e., 1.2 s. Figure 11 shows that this window size is most effective in combination with a 50 Hz sampling frequency.

Input variable selection: There are a total of six input variables to choose from (e, $\dot{e}$, u, $\dot{u}$, x, $\dot{x}$), since there is no fixed *amount* of inputs and the *position* of each input variable does not matter—as is evident from the provided definition of a 1D convolutional forward pass (Figure 8)—this yields a total of 63 unique input variable combinations. Each of these combinations is tested with all other hyperparameters kept at their nominal values (Tables I and II).

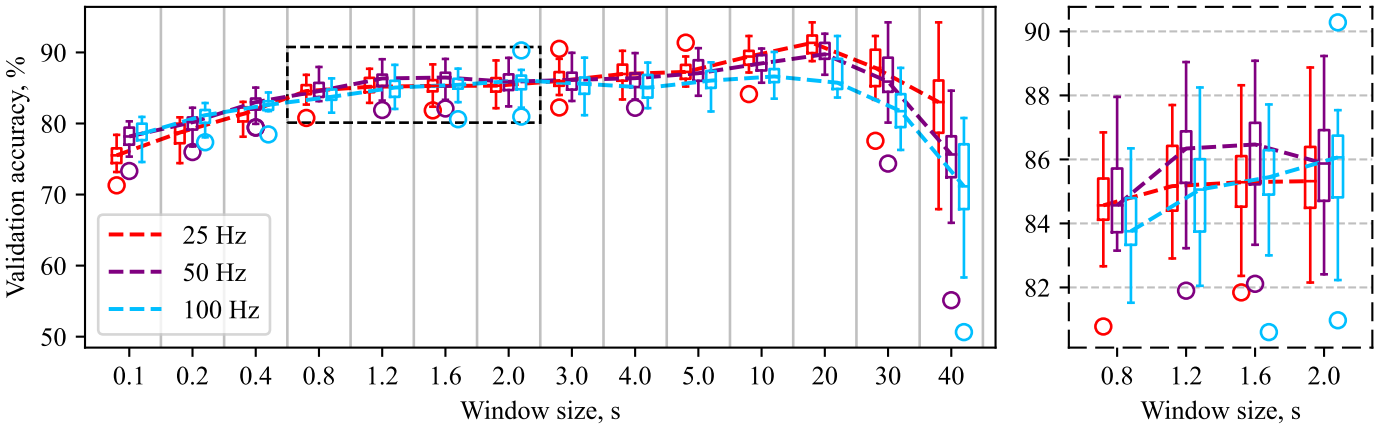


Fig. 11. Classification performance for different combinations of window sizes and sampling frequencies. N = 30, *Data set* = *Data-NM*

The results of the input variable selection optimization are provided in Table III (the complete table is provided in Appendix B-3). The indicated accuracy and loss values are the median value of 30 repeated performance tests. Evidently, the nominal input variable setting of $e + \dot{e} + u + \dot{u}$ is the best performing input combination. Interestingly, this is not the same combination found during optimization by Versteeg [15] and Verkerk [16], confirming that the ideal set of input signals depends on the classification task. All of the input combinations in the top five include the pilot control input u and its time derivative $\dot{u}$, indicating that these two input variables are important to include to achieve good classification performance. When comparing the models that were trained to classify pilot skill level based on only one input variable, it can be seen that $\dot{u}$ is the most effective, indicating that this signal contains the most information for the classifier to distinguish skilled and unskilled behavior. Interestingly, the tracking error e and its time derivative $\dot{e}$ are poor discriminator by themselves, but nevertheless present in the optimal input combination, proving that they are essential when used in combination with u and $\dot{u}$.

TABLE III
RANKED PERFORMANCE OF INPUT COMBINATIONS.

Rank	Input combination	Validation accuracy	Validation loss
1	$\mathbf{u + \dot{u} + e + \dot{e}}$	**86.35%**	**0.3270**
2	$u + \dot{u} + e + \dot{e} + \dot{x}$	84.96%	0.3417
3	$u + \dot{u} + e + \dot{e} + x$	84.71%	0.3486
4	$u + \dot{u} + e + \dot{e} + x + \dot{x}$	83.68%	0.3662
5	$u + \dot{u} + x + \dot{x}$	82.82%	0.3876
⋮	⋮	⋮	⋮
31	$\dot{u}$	76.92%	0.4880
⋮	⋮	⋮	⋮
50	u	73.15%	0.5350
⋮	⋮	⋮	⋮
59	e	65.61%	0.6108
60	x	63.58%	0.6391
61	$\dot{x}$	63.01%	0.6382
62	$\dot{e}$	61.71%	0.6478
63	$\dot{e} + \dot{x}$	58.96%	0.6729

2) Model Hyperparameters: Three essential parameters of the ResNet architecture are optimized to find the best performing ResNet adaptation for time series classification of pilot skill level: the number of residual blocks—or depth of network—d, the number of filters K_i in the convolutional layers of each residual block, and the kernel sizes $\{L_1, L_2, L_3\}$ of the three stacked convolutional layers in each block.

All possible combinations between the following settings are tested in this hyperparameter grid search optimization: four depth settings $d = [1, 2, 3, 4]$, three filter settings $K_1 = [16, 32, 64]$ (only K_1 is set, because the number of filters K_i is doubled at every evenly indexed residual block), and four kernel size settings $\{L_1, L_2, L_3\} = [\{5, 3, 2\}, \{8, 5, 3\}, \{16, 10, 6\}, \{32, 20, 12\}]$. All other hyperparameters are kept at their nominal values (Tables I and II) during this analysis. The results of the hyperparameter optimization are shown in Figure 12. Note that only the first kernel size L_1 is shown to save plotting space. The size of each sphere in Figure 12 indicates the uncertainty in performance outcome when training the classifier with that respective setting. Therefore, the smallest sphere with the brightest blue tone indicates the optimal setting. Evidently, it appears that the baseline model hyperparameters settings ($d = 3$, $K = \{64, 128, 128\}$, $L = \{8, 5, 3\}$) yield the best classification performance. With these model settings, the network consists of 505,986 trainable parameters. To put that into perspective, the smallest tested network ($d = 1$, $K = \{16\}$, $L = \{5, 3, 2\}$) has 1,890 trainable parameters, and the largest network ($d = 4$, $K = \{64, 128, 128, 256\}$, $L = \{32, 20, 12\}$) has 5,168,514 trainable parameters. Training computational load increased with network size, but not linearly: the largest network took 9 times longer to train than the smallest network.

3) Optimization review: The total training time spent during the optimization process (both the data settings and hyperparameters) is **837 h**, consisting of 43 h for the labeling width optimization, 247 h for window size and sampling frequency optimization, 339 h to optimize the selection of input variables, and 208 h for the grid search model hyperparameter optimization.

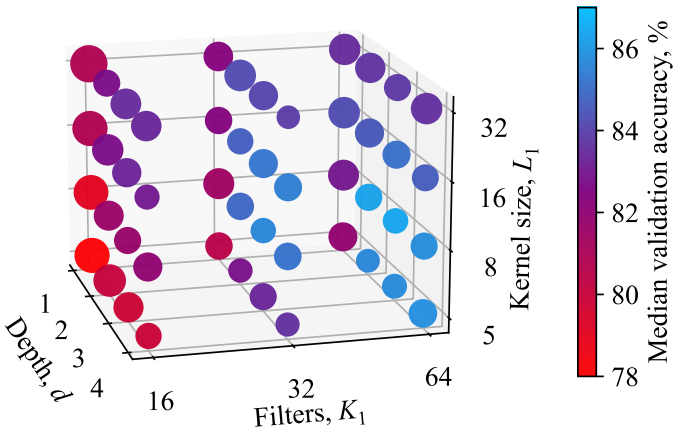


Fig. 12. Hyperparameter grid search optimization results. The size of each sphere indicates the spread in validation accuracy (e.g., like the interquartile range of a boxplot). *Data set = Data-NM*

The final hyperparameters that were found through optimization are presented in Table IV. Evidently, only the labeling width and window size have changed with respect to their original values. Out of those two adjustments, only the labeling width increased the classification performance. The decreased window size resulted in virtually constant classification performance, but the shorter window length was favorable because it allows for faster feedback. When using the optimized hyperparameter settings, an average validation accuracy of **87.95%** is found on the Data-NM (fixed base simulator) set. To test how well these hyperparameter settings transfer to another data set, the optimized settings were also used for training and validating on the Data-M (moving base simulator) set. Interestingly, a validation accuracy of **92.14%** was acquired on the Data-M set, indicating that the tracking data of pilots in a moving-base simulator are more distinguishable—in terms of skill level—than the tracking data of pilots in a fixed-base setting, to the classifier. This is analyzed in more detail in the next subsection.

TABLE IV
OPTIMIZATION RESULTS.

Setting	Symbol	Baseline value	Optimized value
Labeling width	L_w	15 runs	20 runs
Window size	WS	1.6 s	1.2s
Sampling frequency	S_f	50 Hz	50 Hz
Input variables	-	e+$\dot{e}$+u+$\dot{u}$	e+$\dot{e}$+u+$\dot{u}$
Depth	d	3	3
First number of filters	K_1	64	64
First kernel size	L_1	8	8
Data-NM val. accuracy		86.35%	87.95%
Data-M val. accuracy		-	92.14%

B. Performance

The final performance of the classifier is expressed as the average *test* accuracy across all subjects. As established in Section III-C, the test accuracy of one subject is determined by isolating the data of said subject as out-of-sample test set and using the remaining subjects' data as randomly assigned 80% train and 20% validation set. From this analysis it was found that the average test accuracy is **62.36%** for the Data-NM set and **75.13%** for the Data-M set. Evidently, the test accuracy is significantly lower than the previously reported validation accuracy. This discrepancy is explained in the Discussion section. Although the test accuracy may seem to indicate poor classifier performance, the underlying problem may actually be poor class representation of the test data, while the classifier works as intended.

To illustrate this, consider Figure 13, showing the results of the out-of-sample test set analysis for every subject of the Data-NM set. The model output shown on the y-axis is of the skilled neuron, i.e., $P(`skilled')$, which can be seen to generally increase as pilots complete more training runs. Since there are only two output neurons with softmax activation, $P(`unskilled') = 1 - P(`skilled')$, making separate indication of $P(`unskilled')$ redundant. The test accuracy indicated outside of the dashed purple lines indicates the classification accuracy in the respective labeled region. This is defined as the amount of correctly predicted classes (where class prediction is 'skilled' if $P(`skilled') > 0.5$ and vice-versa) divided by the total amount of samples in that region. To offer context to the model skill predictions, the root mean square tracking error data (RMS(e)) of each subject is also indicated. Looking at the evaluation of Subject 7 in Figure 13, the earlier remark about poor class representation of test data immediately becomes apparent: the model almost only predicts 'unskilled' behavior for every tracking run Subject 7 performed, resulting in very high test accuracy in the 'unskilled' label region (91.2%), but very poor test accuracy in the 'skilled' label region (5.4%). Though this may seem like poor classification, the RMS(e) (red line) actually shows that Subject 7 did indeed always have above average RMS(e) (dashed red line) throughout the entire training cycle, indicating comparatively poor tracking performance. The exact opposite effect can be observed for Subject 1, who starts out with better than average tracking performance (in terms of RMS(e)) and is therefore predominantly classified as 'skilled', resulting in low test accuracy in the starting tracking runs and high test accuracy in the last tracking runs. This interaction between model output and RMS(e) can also be observed for sudden changes in control performance between different runs, for example whenever Subject 2's RMS(e) peaks, the model output lowers.

To further explore the exchange between model prediction and subject tracking performance, examine Figure 14. This figure can be seen as an aggregation of Figure 13 where all subject tracking performances of the data-NM set are combined in one figure. Now every marker indicates an individual tracking run, and the respective color indicates the average model output between all samples (1.2 s windows) in said run. Additionally, the y-axis shows the RMS(e) of each individual run as a measure of tracking performance. The bottom graph shows the average model output over a range of run indices, clearly displaying the poor classification accuracy, especially for runs of experienced individuals with high tracking error. From the top graph it can be observed that, even though an *experience*-based labeling method was used, there is a strong

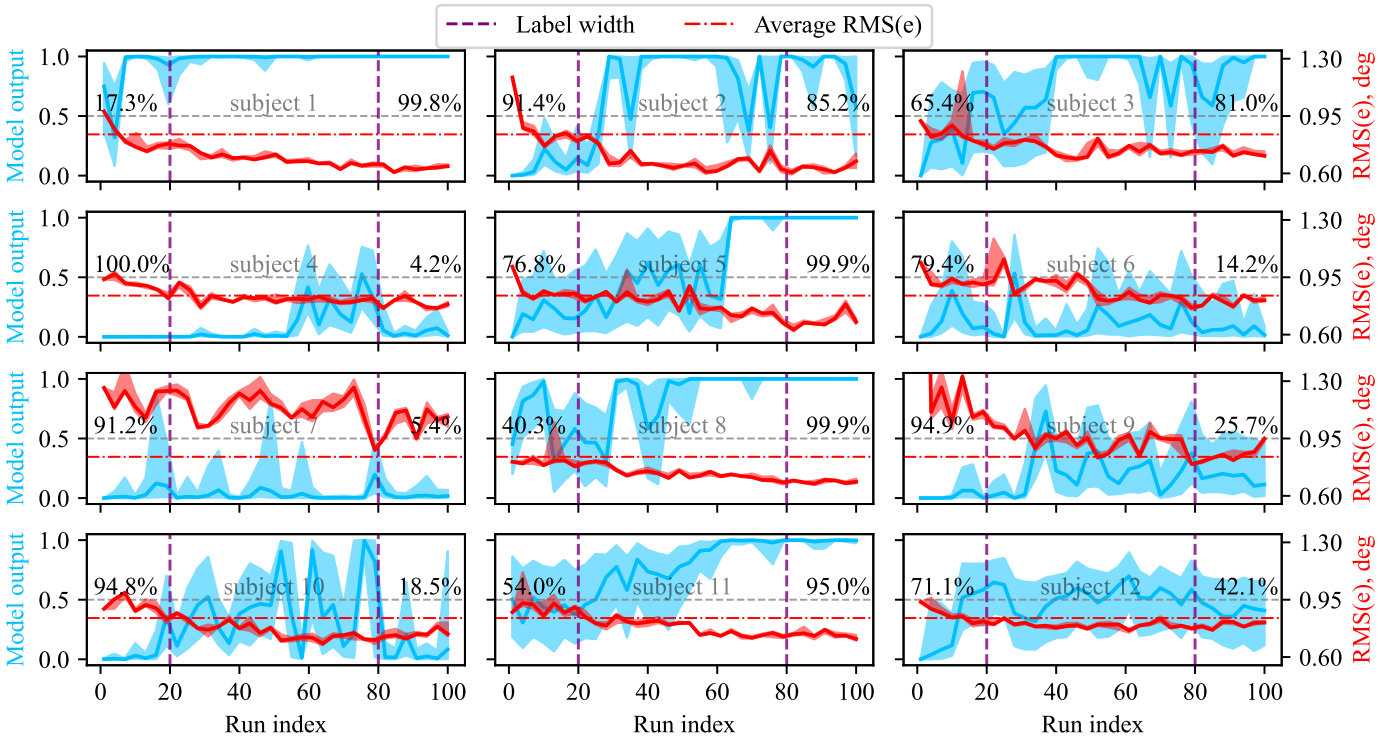


Fig. 13. Results of subject as out-of-sample test set analysis. The blue and red line display the median model output and RMS(e) per bin of three runs, respectively, with the shaded areas indicating the corresponding interquartile range. Displayed percentages show test accuracy in labeled region. *Data set = Data-NM*

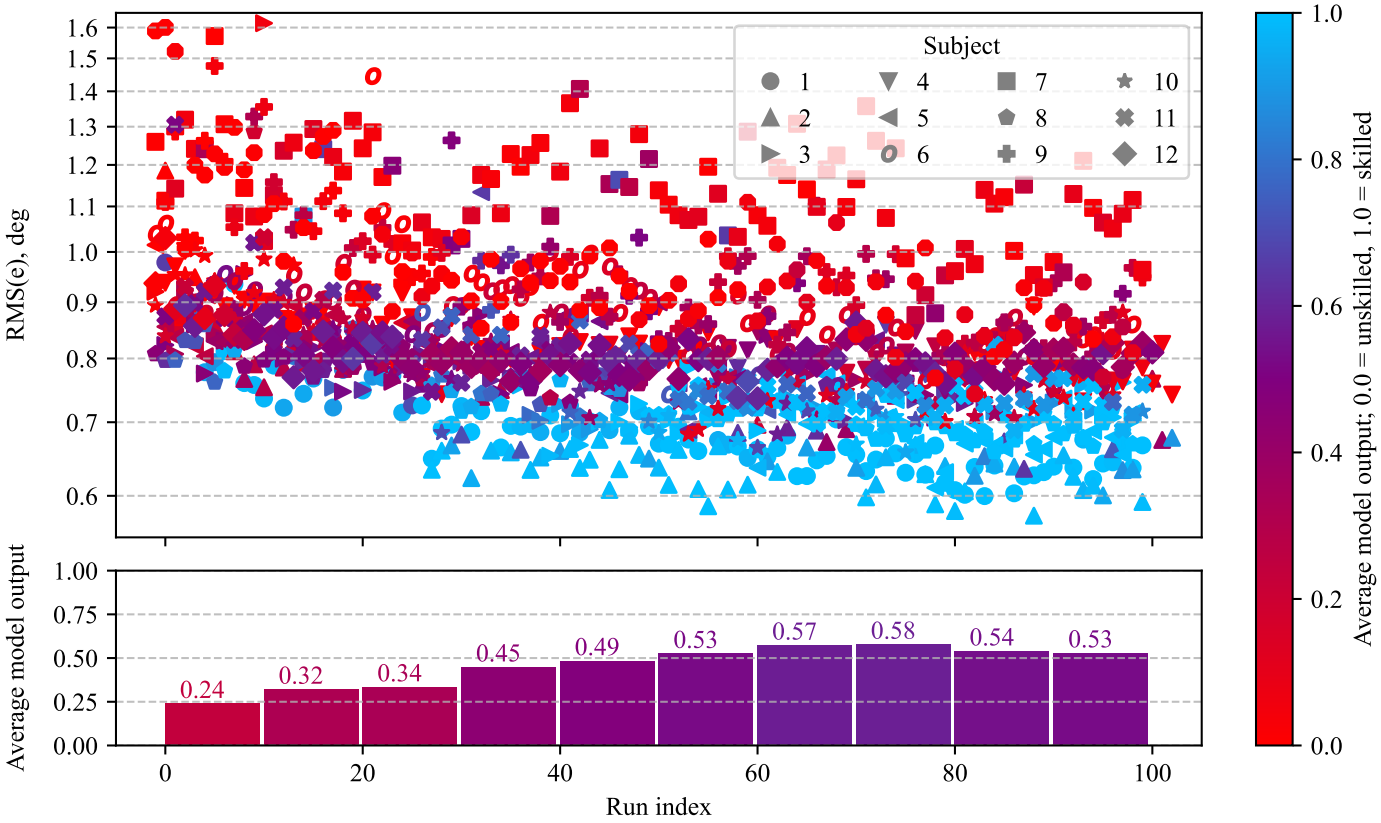


Fig. 14. Every marker in the top figure represents an individual tracking run of one subject, the color of the marker indicates the average model output of said run. Bottom bar chart displays average model output for bins of run indices. *Data set = Data-NM*

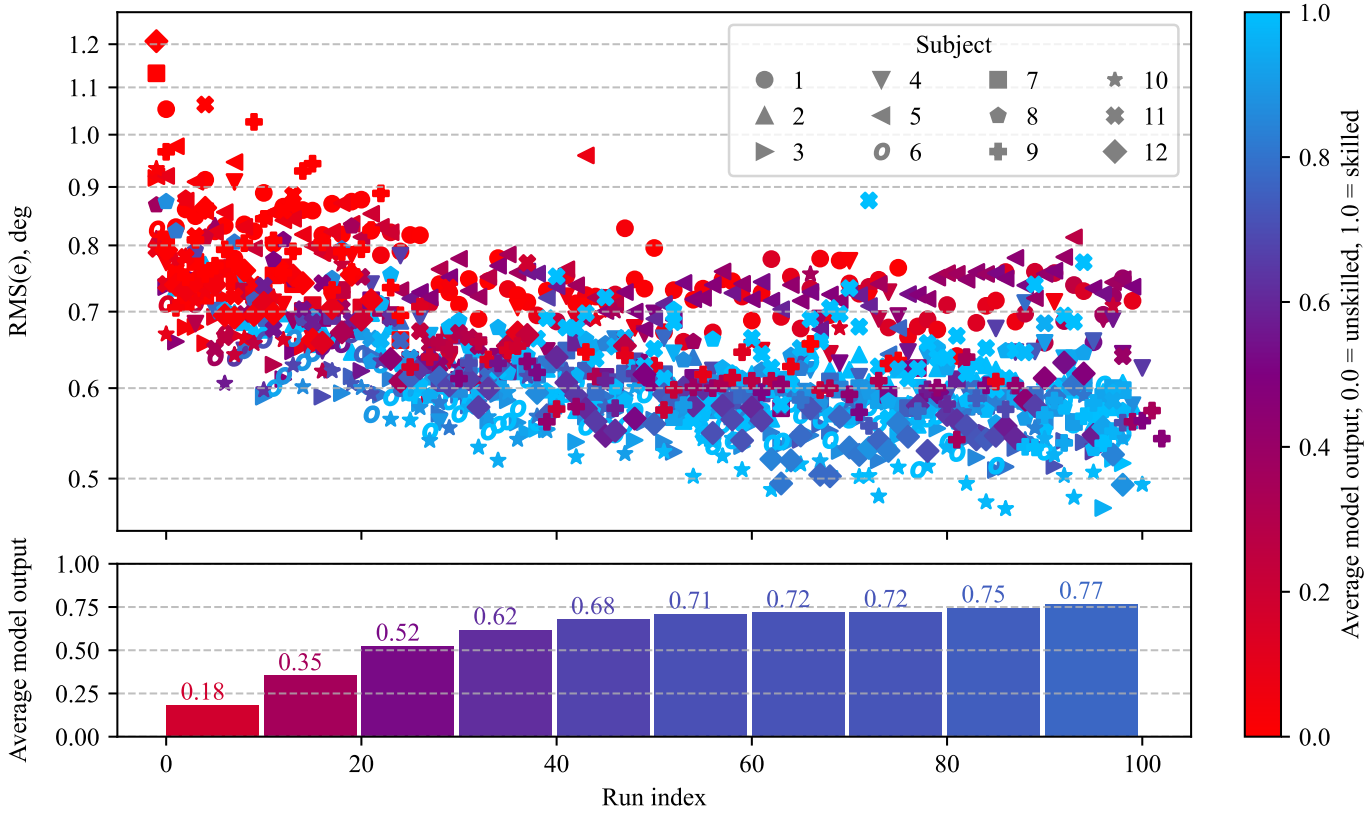


Fig. 15. Every marker in the top figure represents an individual tracking run of one subject, the color of the marker indicates the average model output of said run. Bottom bar chart displays average model output for bins of run indices. *Data set = Data-M*

correlation between tracking performance and model output. Namely, the shift from 'unskilled' (red) to 'skilled' (blue) is more visible from top to bottom than from left to right. This makes it seem as though the deep learning model bases its skill level predictions on just the tracking error, but a closer look at model output of—for example—Subject 10 indicates that there is more at play. Subject 10 is, despite having a relatively low RMS(e), often classified as 'unskilled', contradicting the previous observation. On a side note, extremely low classification accuracy—like 4.2% and 5.4% of the 'skilled' behavior of Subjects 4 and 7, respectively—does indicate that the classifier is very certain in its predictions (e.g., 50% accuracy could indicate the classifier is simply guessing). This observation strengthens the hypothesis that the problem lies in the labeling of the data (i.e., the task definition) and not in the predictions of the classifier.

As an additional evaluation step, the aggregated results of the test performance analysis on the Data-M set are displayed in Figure 15. It is immediately apparent that classifier shows more desired behavior; the bottom graph now nicely visualizes the gradual increase in average skill level between the subjects during training. This improved classification performance can partially be explained by the more consistent learning curve that is visible in the RMS(e) values of the Data-M set, compared to the Data-NM set. However, there are some subjects (e.g., Subject 11) who are classified as 'skilled' even though they have a relatively high RMS(e). Evidently, the deep learning model derives its predictions from more than the mere tracking error, the following section will attempt to shed light on how the model output is established.

C. Explainability

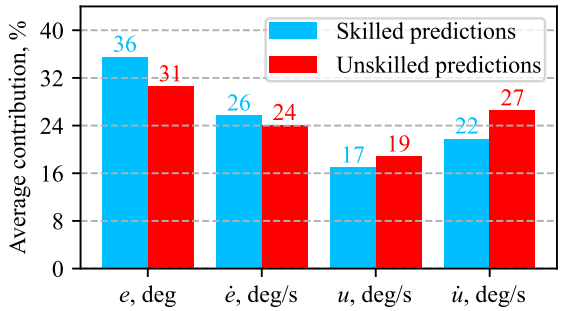


Fig. 16. Global variable importance—expressed as average contribution to model output—for samples predicted as either 'skilled' or 'unskilled'. *Data set = Data-NM*

1) SHAP: Here SHAP (SHapley Additive exPlanations) [25] is used to add transparency to the trained classifier. As was visualized in Figure 9, SHAP can provide the marginal contributions of each input variable to a single class prediction of the deep learning model, which is referred to as a *local* explanation. Combining a very large set of local

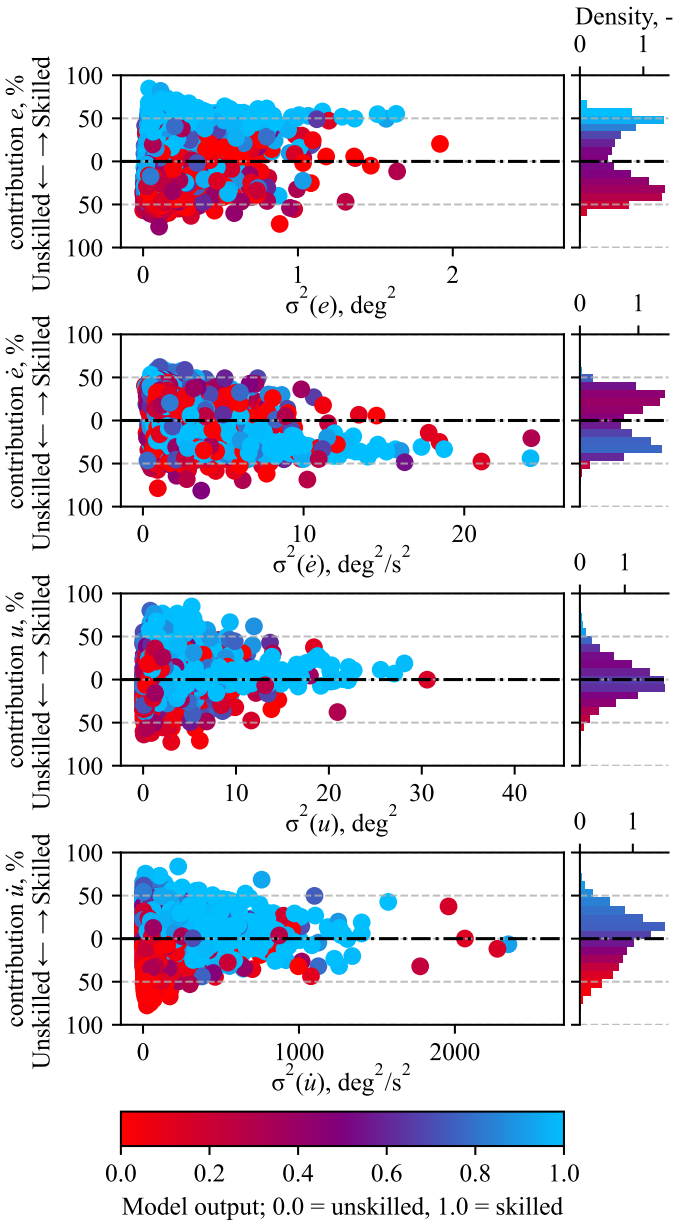


Fig. 17. Variance of input variables versus their respective contribution to the class prediction. Each dot indicates one input sample X, with its color reflecting the corresponding model output $f(X)$. Horizontal histograms indicate probability density of output contribution. *Data set = Data-NM*

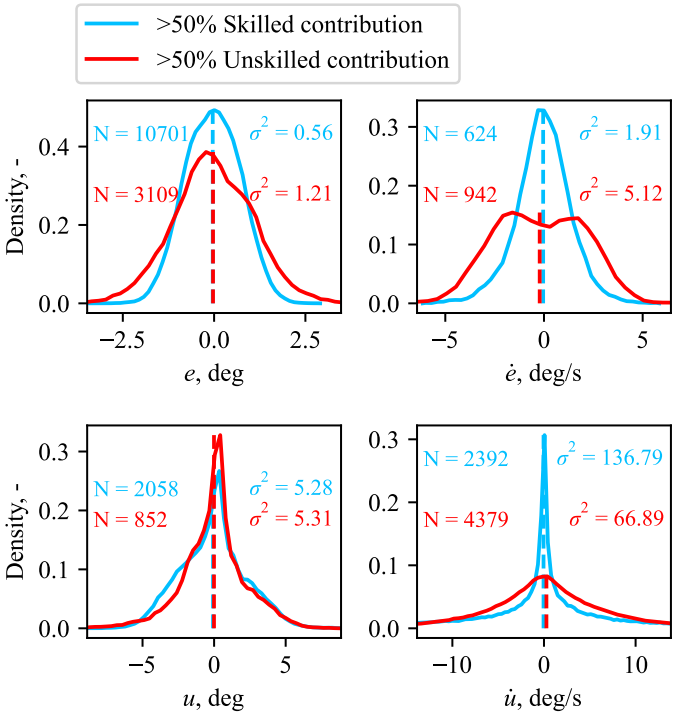


Fig. 18. Probability density function (PDF) of input signals with more than 50% contribution to the model output. N indicates the amount of samples (after filtering minimum contribution) per PDF, out of 88,000 total samples. *Data set = Data-NM*

explanations may offer perspective on the overall working of the deep learning model, i.e., providing a *global* explanation [76]. The outcome of this analysis is shown in Figure 16. Here 88,000 non-overlapping samples were taken from the Data-NM set, and for each sample the local input variable importance (expressed as an absolute percentage-wise contribution to the model output) was calculated using SHAP. Averaging the contribution of each input variable over the 88,000 predictions—and distinguishing between 'skilled' and 'unskilled' predictions—provides the global variable importance as presented in Figure 16. Noticeably, e is the most important (highest average contribution) to determine model output for both skilled and unskilled predictions, while u is the least important (lowest average contribution). As is further discussed in the Discussion, this appears inconsistent with the variable importance found during hyperparameter optimization (Table III). It can also be observed that—according to the classifier—e and $\dot{e}$ are stronger discriminators for skilled behavior than for unskilled behavior, whereas u and $\dot{u}$ on the other hand appear to have a stronger contribution in unskilled predictions.

To develop a better understanding of what characteristics drive an input signal to have a high—or low—contribution to the model output, a comparison is made between the variance of each input signal and their resulting percentage-wise contribution. The results of this analysis are shown in Figure 17, where every dot represents a single sample X with its color reflecting the corresponding class prediction $f(X)$. The vertical position of each dot indicates the marginal contribution of the respective input variable to the model output (see Figure 9 for the detailed definition), and the horizontal position indicates the variance σ^2 of said input variable in sample X. Note that the probability density functions shown on the right side of each graph reinforce the findings of Figure 16, as the more dense the probability is away from 0% contribution, the more important that input variable is globally. What is interesting is that e and $\dot{e}$ appear to have either low- or high contribution, whereas u and $\dot{u}$ show a more balanced distribution around 0% contribution. Also observe that $\dot{e}$ often contradicts the final model output, as approximately 24,000 confident skilled

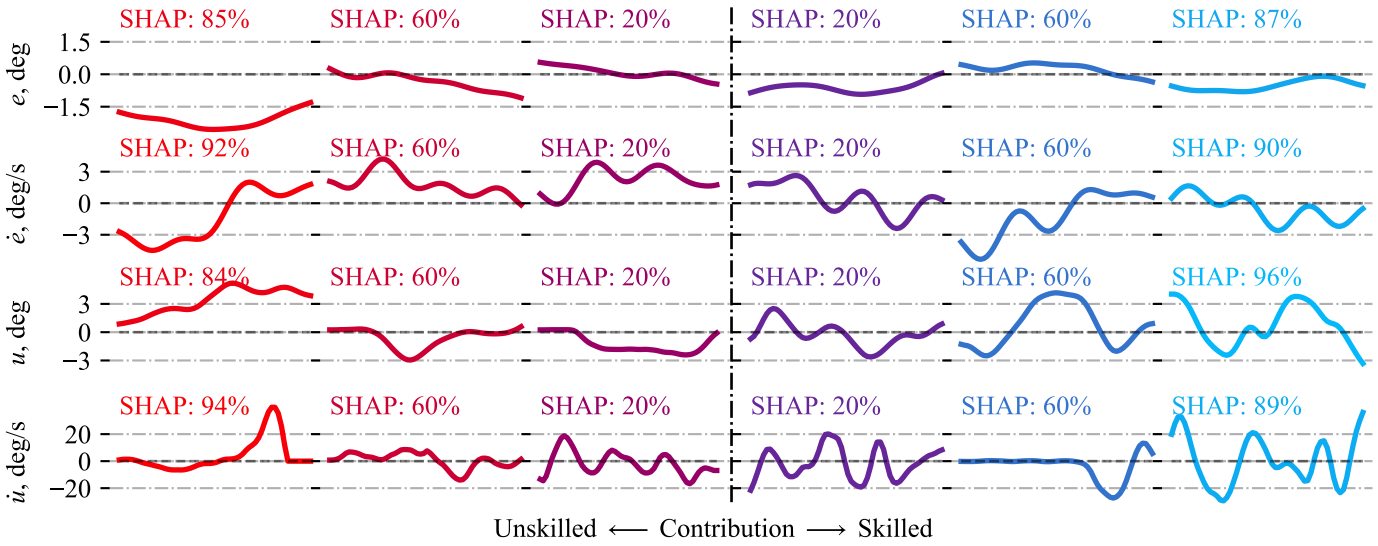


Fig. 19. Examples of input signals with different ranges of % contribution towards either skilled or unskilled predictions. *Data set = Data-NM*

predictions with $f(X) > 0.8$ (blue dots) occur even when $\dot{e}$ has a contribution towards unskilled behavior (out of the 88,000 total explained predictions).

Figure 17 also explores whether there is a correlation between the variance of input variables and their contribution to class predictions, i.e., whether a high or low signal variance goes hand in hand with a high or low contribution. From Figure 17 it can be concluded that such a correlation is practically absent, there is only a weak correlation between high $\sigma^2(e)$ and skilled contribution and a slightly stronger correlation between high $\sigma^2(\dot{e})$ and unskilled contribution. For $\sigma^2(u)$ and $\sigma^2(\dot{u})$ no such correlation can be distinguished at all. This is an indication that the deep learning model extracts features from the raw time series data that are more particular than, for example, the variance. In further analyses (not included here), the same was found to be true for the mean, minimum, and maximum of the signals.

To get a better sense of what features the classifier might be picking up, a comparison is made between the probability density functions (PDFs) of input signals with a high contribution ($> 50\%$) towards skilled predictions and the PDFs of inputs with a high contribution ($> 50\%$) towards unskilled predictions. These PDFs are visualized in Figure 18. From this figure it becomes apparent that only the PDFs of e, $\dot{e}$, and $\dot{u}$ show different distributions between skilled and unskilled contributions. Specifically, it appears that control behavior signals with a high probability density value around 0 deg (or deg/s) contribute towards skilled predictions. This observation is supported by the sample signals shown in Figure 19. Indeed the signals with the highest contribution towards unskilled predictions (red) are located further away from 0 deg (or deg/s) than signals with contributions towards skilled predictions (blue).

2) Activation Maximization: Although the sample signals shown in Figure 19 offer some insight into how 'unskilled' and 'skilled' signals differ, it remains difficult to clearly recognize any discriminative patterns. Perhaps distinctive features could be identified if a larger set of samples pertaining to each class was investigated, but it may be more effective to reveal the trained deep learning classifier's internal representation of each class. Activation maximization [26, 70] is used to retrieve the classifier's perception of the classes it has learned to distinguish. The results of this class visualization method for the Data-NM set are shown in Figure 20. Classifiers trained with either Data-NM or Data-M produced similar outcomes. As a verification step, these generated samples were fed back to the classification model and indeed gave maximum probability output for their target prediction. Notice how a 'skilled' sample appears to be more smooth, whereas an 'unskilled' sample seems more variable and noisy. A possible explanation for this is the stronger *remnant* that is expected to be present in unskilled pilot control behavior.

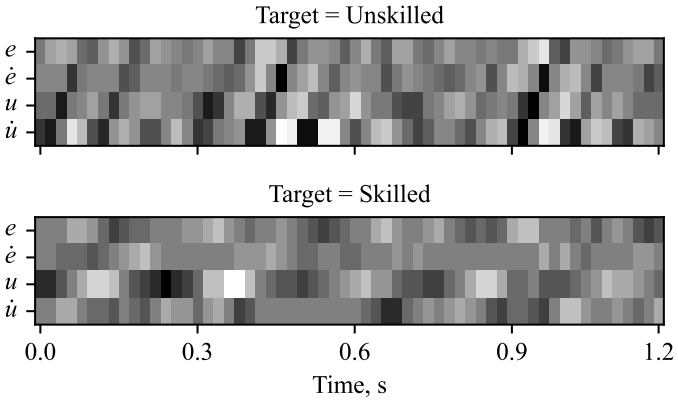


Fig. 20. Class visualizations generated using activation maximization. *Data set = Data-NM*

Remnant is defined as all control behavior that is not linearly correlated with the target and disturbance signals of the tracking task [77–79]. It predominantly consists of 1) pure noise injected by the human operator, 2) nonlinear control

operations such as perception thresholds and control rate saturation, and 3) nonsteady and time-varying control behavior [77]. All of which are effects that are more likely present in inexperienced control behavior [23]. If it is the case that the classifier recognizes remnant in control behavior, then that is a very promising capability as it usually takes much longer time traces to effectively identify remnant [80]. Appendix A further explores the effect of remnant on the classifier performance.

VI. Discussion

This paper set out to implement—and optimize—a deep learning approach to identify skill level of pilots based on short windows of time recordings of human control behavior. Additionally, an effort was made to add transparency to the deep learning model by making the results interpretable. The result of this work is an optimized ResNet architecture adapted for time series classification that can identify pilot skill level based on 1.2 s windows of tracking data with 92% validation accuracy in moving-base simulations and 88% validation accuracy in fixed-base simulations. When simulating the real life application of the trained classifier on a separate test set, the classification accuracy drops to 62% and 75% for the data sets with- and without motion feedback, respectively. The following paragraphs will reflect on the results presented in this paper and discuss their relevance, shortcomings, and potential improvements.

The first and foremost item to be discussed is the labeling of the control data, as the effects of this key operation trickle down into every other aspect of this research. Namely, the definition by which the data are labeled determines: 1) what it is that the trained deep learning classifier will be good at classifying (e.g., labeling by low/high RMS(e) will train to network to recognize low/high RMS(e), an example of this analysis is provided in Appendix C), 2) which hyperparameters are found during optimization of the particular job (e.g., different input variables may be more important for the identification of different class definitions [15, 16]), and 3) the local- and global explanations of the classifier predictions (e.g., a network trained to recognize low/high RMS(e) will probably emphasize the importance of input e more). Evidently, the labeling method is critical and completely shapes the outcome of the trained classifier. In this research, the decision was made to label the pilot skill level by the amount of experience the pilots had in the appointed control task, labeling each subject's first 20 runs as 'unskilled', and their last 20 runs as 'skilled'. This decision determined that the desired behavior of the trained classifier is to effectively recognize whether a pilot is inexperienced (at the beginning of their training) or experienced (at the end of their training). What is difficult about this definition is that every pilot learns at a different rate and starts at a different initial skill level. For example, one pilot may go from a 4/10 at the beginning of their training to a 6/10 at the end of their training, whereas another may go from a 6/10 to a 10/10, yet both will be classified the same (an example of this can be seen when comparing Subject 7 and Subject 1 in Figure 13). This problem is amplified when considering that the human operator is a time-varying system [14], thus every observed time window is not an equally good representation of its respective class as another. This data *mislabeling* deteriorates the resulting classification accuracy both during training, when the classifier is taught the wrong label, and during testing, when the trained classifier is penalized for "wrongly" classifying a sample with an incorrect label. The latter of these two effects has a stronger effect on the classification performance than the first, as deep learning classifiers have been shown to be more resilient towards label noise in the train set [81].

From the input data settings optimization, it was concluded that 1.2 s windows sampled at 50 Hz are optimal. This followed from a strategy to keep the window size short, so that—when implementing the classifier as a real-time skill level detector—direct feedback could be provided. This is desirable if, for example, a skill deterioration warning must be given. However, it could be seen that short window sizes came at a cost in terms of validation accuracy, as window sizes in the 10-20 second range provided a performance improvement of up to 6%. By doing skill level predictions every second, based on the control behavior of a larger window (e.g., ten seconds), the benefit of improved classification performance with longer window sizes could be adopted to provide direct feedback. This would result in a sort of "moving average" skill level prediction over the last ten seconds—of higher accuracy, but lower sensitivity to brief changes in the control behavior—that is still readily available at every second for direct feedback implementations such as skill deterioration warnings and scalable autonomy of the autopilot. It must also be investigated whether the same effect can be accomplished (perhaps with greater accuracy) by still using a short window size, but using a majority vote [38] between samples within a specified time frame, or by utilizing a Markov chain [82] between consecutive samples.

Currently, the input variable selection optimization was done on the fixed-base data set (Data-NM), and the resulting 'optimal' selection of input variables was used for both the performance analyses of Data-NM and Data-M. It could be that classification of the time traces of subjects in the moving base setting (Data-M) may be more accurate with a different set of input variables. Namely, the subjects in Data-M received information of the current aircraft state x by means of motion feedback, unlike the subjects in Data-NM who were only provided with e through a compensatory display. The training/testing of a classifier tasked to predict the skill level of subjects in Data-M may benefit from additional inputs x (and $\dot{x}$) more than a classifier tasked to predict the skill level of subjects in Data-NM, as the deep learning model could learn to recognize patterns such as reaction time between the motion feedback and the control input of the pilot. Verkerk [16] describes such a phenomenon, where introduction of x as an additional input parameter—to a classifier identifying display types based on manual control behavior—resulted in an increased accuracy. He hypothesized that the synergy between x and e allowed the classifier to

recognize effective time delay in the pilot response, which is a distinctive characteristic that distinguishes the control strategy pertaining to different display types. It must be noted, however, that potential input parameters should be selected carefully. For example, Versteeg [15] showed that including both u and x as inputs to a controlled-element-dynamics-classifier produced perfectly accurate results. However, these predictions were *trivial*, as the classifier could now directly identify the controlled element dynamics as the link between u and x.

During the model-specific hyperparameter optimization, it was found that the nominal settings resulted in the best classification performance. This optimization was done by testing every combination of predetermined sets of hyperparameter values from a list (i.e., grid search), resulting in network depth $d = 3$, number of filters $K = 64, 128, 128$ across the residual blocks, and kernel sizes $L = 8, 5, 3$ in the convolutional layers of each residual block. However, this optimization could be done more efficiently (and potentially reap better results, because of the more effectively utilized training time) by using randomly searching hyperparameter values over the domain [83].

As reported in the Results section, the discrepancy between validation accuracy and test accuracy is found to be roughly 20% for both the Data-NM and the Data-M set. The explanation for the significantly higher validation accuracy is twofold. First, the validation accuracy is inherently expected to be higher than the test accuracy, as the deep learning model was specifically instructed to only save its weights at minimum validation loss. This means that, although the model is trained on separate training data, it is still optimized to perform the best on the validation data. Second, both the training and the validation data contain examples of the *same* subjects that are a nonconformity to the expected learning curve (i.e., wrongly labeled), whereas the test data are of a completely isolated separate subject. This second effect is especially detrimental to the test performance. To illustrate this further, consider again Subject7 in Figure 13. The reason the classifier has such a low classification accuracy on the 'skilled' portion of Subject7's control behavior (5.4%) is that the classifier was never trained to understand that this subject's poor tracking performance—in terms of RMS(e)—in the last twenty runs should be classified as 'skilled'. Contrarily, the train and validation data both *do* contain the *same* subjects and thus the classifier can accurately predict the class of a nonconformity like Subject7. Therefore, the low test accuracy could be resolved by randomly splitting the data of *all* subjects into a—for example—70% train, 20% validation, 10% test set. However, then the resulting test accuracy would not be a realistic representation of the expected classification accuracy in a real world application where the classifier has to predict to skill level of a subject it has never seen before. A true solution lies in expanding the data sets to a contain a larger amount of subjects, so that the classifier is familiar with a wider range of pilots and has thus better generalization. Additionally, the subject's class labels (skill level) could be based on the assessment of a professional flight instructor, so that different learning curves are accounted for and more consistent class representations will be present in the training data. The data set could also be expanded—with more consistent class representations—by simulating additional training data using existing human manual control models. Such an approach to data augmentation with a cybernetic pilot model is discussed in Appendix A. Although the results of this technique are promising, it is not yet beneficial to the accuracy of the classifier.

Using SHapley Additive exPlanations (SHAP) [25], an eXplainable Artificial Intelligence (XAI) method, the input feature importance was computed. This analysis showed that, on average, the importance of the input variables are ranked in the following order: e, $\dot{e}$, $\dot{u}$, u. Interestingly, e and $\dot{e}$ are more important in 'skilled' predictions than in 'unskilled' predictions, whereas u and $\dot{u}$ are more important in 'unskilled' predictions than in 'skilled' predictions. This could signify that the classifier more heavily relies on the remnant in u and $\dot{u}$ for 'unskilled' predictions—which is known to be higher for untrained pilots than for trained pilots [23]—and switches its focus to e and $\dot{e}$ for 'skilled' predictions (where less remnant is present). Activation Maximization offered some more perspective on the classifier's internal representation of 'skilled' and 'unskilled' behavior, displaying a smooth synthetic input as 'skilled' behavior and a more variable, noisy, synthetic input for the 'unskilled' behavior. This further substantiates the hypothesis that the classifier recognizes the level of remnant in the pilot behavior. These results are very promising, as remnant identification usually requires much longer time series [80] (Appendix A further explores the effect of remnant on the deep learning model behavior). However, it is difficult to verify the exact nature of the different class visualizations, as the images are quite abstract and the representative features could have different sources. Potentially, clearer class visualizations could be produced by using DGN-AM [84]: an improved activation maximization (AM) method that uses a deep generator network (DGN) to produce the synthetic images. Also, further research could be done to develop deep learning models that can perform semantic segmentation [85, 86] to isolate the remnant portion of the control behavior. For example, Jansson et al. [87] successfully adopted U-Net [85] to decompose a music audio signal into its vocal and backing track components. Such an approach can potentially enable new insights into the nonlinear phenomenon of human pilot remnant.

The advantage of using SHAP is that it is a model-agnostic explainability technique [24], meaning that it can be utilized on *any* model with the intent to extract information of the inner workings of that model. The disadvantage, however, is that it requires a simplified explanation model [25] to interpret the main model, which may make it less and accurate and more computationally expensive. Model-specific explainability methods available to convolutional neural networks—like CAM [63] or Grad-CAM [64]—may produce more reliable explanations and require less computation time, as they directly tap into the trained network parameters to compute the local feature importance. However, a shortcoming of using 1D

convolutional layers (as used in this paper) is that the aforementioned class activation mapping methods also produce 1D explanations; only the temporal importance is highlighted, indicating what segment of time led to the class prediction, but the individual contribution of each input variable is missing (this phenomenom is visualized in Appendix D-1). Using 2D convolutional layers would enable the identification of individual input contributions, but it was empirically found that using 2D layers throughout the ResNet network degraded classification performance by 7% (results shown in Appendix B-4) at the additional cost of five times longer training times due to the larger amount of trainable parameters in 2D layers vs 1D layers (accumulating to 2,859,074 total trainable parameters instead of 505,986). A potential solution that could be researched is the utilization of a network that uses both 2D and 1D convolutional layers in parallel [88] or in sequence [89], to preserve both the temporal and spatial dynamics of multivariate time series input, so that the individual contribution of each input channel can be determined.

A last note to be made about the XAI results is that the discrepancy in variable importance between the SHAP analysis (Figure 16) and the input variable selection optimization (Table III)—i.e., the global explaination by SHAP found e as most important and u as least important, whereas the input variable optimization identified u as more important than e—does not necessarily mean that SHAP has produced inaccurate results. For example, when only a single input variable is provided to the classifier, u has proven to be a better indicator of pilot skill level than e. However, when both u and e are provided, the largest contributor to the model output could be e, as—in combination with u—it potentially enables the classifier to identify the effective average pilot response delay.

To conclude this discussion, some final remarks are made about the applicability of the proposed method. Overall the results of this research are promising. The achieved 92% validation accuracy on the Data-M set with just 1.2 s windows shows how potent this deep learning classification method is. However, the significant drop in accuracy, when testing this one-size-fits-all skill level identification method on isolated subjects, shows that the classifier—in its current state—is not good at predicting individual skill development. A larger set of subjects' tracking data, better (manual) labeling of the training data, and further optimization of the deep learning architecture may help to overcome this problem. Also, currently the classifier has only been trained and tested on data of pilots performing a single axis *compensatory* tracking task. More research should be done about the applicability of the proposed skill level classification method on different tracking tasks and display types. To get closer to a real life implementation of the proposed skill evaluation system, a similar training of manual control skill experiment should be conducted in a high-fidelity, multi-axis, moving-base flying simulation. Furthermore, the trade-off between prediction accuracy and sensitivity to quick control adaptations should be optimized for smooth interaction between the human operator and the automated aircraft.

VII. Conclusion

This paper introduces a deep learning approach for quantifying pilot skill level in the time-domain. Applied to data from a recent simulator training experiment, in which fully task-naive participants were trained to perform a pitch tracking task, this method was found to be able to effectively classify the participant's skill level as either 'unskilled' (at the beginning of training) or 'skilled' (at the end of training) based on only 1.2 seconds of control data. Using a ResNet CNN architecture, an average validation accuracy of up to 92% could be achieved, but this accuracy significantly dropped when testing the trained classifier on isolated subjects with off-nominal learning curves. This drop in accuracy signifies that this one-size-fits-all method—in its current state—is not always effective at identifying individual skill development. During classifier optimization it was discovered that a combination of the pilot control input u, the tracking error e, and their first order time derivatives $\dot{u}$ and $\dot{e}$ serve as the best discriminators for 'skilled' and 'unskilled' pilot behavior.

Deploying SHAP, it was discovered that e is the most important feature, followed by $\dot{e}$ for 'skilled' predictions and $\dot{u}$ for 'unskilled' predictions. Surprisingly, u was found to be the least important feature. An activation maximization visualization technique of the model's internal representations of each skill level revealed that the trained classifier may recognize remnant in the unskilled pilot control behavior.

This explainable deep learning approach to skill level identification sets the first step towards online quantitative evaluation of control behavior, opening a new realm of possibilities to enhance safety in automated systems that rely on smooth interaction with the human operator.

References

[1] N. A. Stanton and M. S. Young, "Vehicle automation and driving performance," *Ergonomics*, vol. 41, no. 7, pp. 1014–1028, 1998.

[2] A. Calvi, F. D'Amico, L. B. Ciampoli, and C. Ferrante, "Evaluation of driving performance after a transition from automated to manual control: a driving simulator study," *Transportation Research Procedia*, vol. 45, pp. 755–762, 2020, transport Infrastructure and systems in a changing world. Towards a more sustainable, reliable and smarter mobility.TIS Roma 2019 Conference Proceedings.

[3] J. C. F. de Winter, R. Happee, M. H. Martens, and N. A. Stanton, "Effects of adaptive cruise control and highly automated driving on workload and situation awareness: A review of the empirical evidence," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 27, pp. 196–217, 2014, vehicle Automation and Driver Behaviour.

[4] S. M. Casner, R. W. Geven, M. P. Recker, and J. W. Schooler, "The retention of manual flying skills in the automated cockpit," *Human Factors*, vol. 56, no. 8, pp. 1506–1516, 2014, pMID: 25509828.

[5] Federal Aviation Administration, "Enhanced FAA oversight could reduce hazards associated with increased use of flight deck automation." FAA, Audit Report AV-2016-013, 2016.

[6] International Air Transport Association, "Aircraft handling and manual flying skills," IATA, Report, 2020.

[7] D. M. Pool and P. M. T. Zaal, "A cybernetic approach to assess the training of manual control skills," *IFAC-PapersOnLine*, vol. 49, no. 19, pp. 343–348, 2016, 13th IFAC Symposium on Analysis, Design, and Evaluation of Human-Machine Systems HMS 2016.

[8] D. M. Pool, G. A. Harder, and M. M. van Paassen, "Effects of simulator motion feedback on training of skill-based control behavior," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 4, pp. 889–902, 2016.

[9] M. Mulder, D. M. Pool, D. A. Abbink, E. R. Boer, and M. M. van Paassen, "Fundamental issues in manual control cybernetics," *IFAC-PapersOnLine*, vol. 49, no. 19, pp. 1–6, 2016, 13th IFAC Symposium on Analysis, Design, and Evaluation ofHuman-Machine Systems HMS 2016.

[10] P. Zaal and B. Sweet, "Estimation of time-varying pilot model parameters," in *AIAA Modeling and Simulation Technologies Conference*, 2011.

[11] R. F. M. Duarte, D. M. Pool, M. M. van Paassen, and M. Mulder, "Experimental scheduling functions for global LPV human controller modeling," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 853–15 858, 2017, 20th IFAC World Congress.

[12] A. Popovici, P. Zaal, and D. M. Pool, "Dual extended kalman filter for the identification of time-varying human manual control behavior," in *AIAA Modeling and Simulation Technologies Conference*, 2017.

[13] J. Rojer, D. M. Pool, M. M. van Paassen, and M. Mulder, "UKF-based identification of time-varying manual control behaviour," *IFAC-PapersOnLine*, vol. 52, no. 19, pp. 109–114, 2019, 14th IFAC Symposium on Analysis, Design, and Evaluation of Human Machine Systems HMS 2019.

[14] M. Mulder, D. M. Pool, D. A. Abbink, E. R. Boer, P. M. T. Zaal, F. M. Drop, K. van der El, and M. M. van Paassen, "Manual control cybernetics: State-of-the-art and current trends," *IEEE Transactions on Human-Machine Systems*, vol. 48, no. 5, pp. 468–485, 2018.

[15] R. Versteeg, "Classifying human control behavior by artificial intelligence," MSc. thesis, Delft University of Technology, Faculty of Aerospace Engineering, 2019, unpublished. [Online]. Available: http://repository.tudelft.nl

[16] G. J. H. A. Verkerk, "Classifying human manual control behavior in tracking tasks with various display types using the InceptionTime CNN," MSc. thesis, Delft University of Technology, Faculty of Aerospace Engineering, 2021, unpublished. [Online]. Available: http://repository.tudelft.nl

[17] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, Mar 2019.

[18] A. Jain, A. Singh, H. S. Koppula, S. Soh, and A. Saxena, "Recurrent neural networks for driver activity anticipation via sensory-fusion architecture," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3118–3125.

[19] K. Saleh, M. Hossny, and S. Nahavandi, "Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.

[20] F. Tango and M. Botta, "Real-time detection system of driver distraction using machine learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 894–905, 2013.

[21] S. K. R. Nittala, C. P. Elkin, J. M. Kiker, R. Meyer, J. Curro, A. K. Reiter, K. S. Xu, and V. K. Devabhaktuni, "Pilot skill level and workload prediction for sliding-scale autonomy," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 1166–1173.

[22] P. Xi, A. Law, R. Goubran, and C. Shu, "Pilot workload prediction from ECG using deep convolutional neural networks," in *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, 2019, pp. 1–6.

[23] R. Wijlens, P. Zaal, and D. Pool, "Retention of manual control skills in multi-axis tracking tasks," in *AIAA Scitech 2020 Forum*, 01 2020.

[24] A. B. Arrieta, N. Díaz-Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," 2019, arXiv: 1910.10045.

[25] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[26] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2014, arXiv: 1312.6034.

[27] P. M. T. Zaal, D. M. Pool, J. de Bruin, M. Mulder, and M. M. van Paassen, "Use of pitch and heave motion cues in a pitch control task," *Journal of Guidance, Control, and Dynamics*, vol. 32, no. 2, pp. 366–377, 2009.

[28] R. L. Stapleford, R. A. Peters, and F. R. Alex, "Experiments and a model for pilot dynamics with visual and motion inputs," National Aeronautics and Space Administration, Contract. Rep. NASA CR-1325, May 1969.

[29] D. M. Pool, P. M. T. Zaal, M. M. van Paassen, and

M. Mulder, "Effects of heave washout settings in aircraft pitch disturbance rejection," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 1, pp. 29–41, 2010.

[30] F. M. Nieuwenhuizen, M. Mulder, M. M. van Paassen, and H. H. Bülthoff, "Influences of simulator motion system characteristics on pilot control behavior," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 3, pp. 667–676, 2013.

[31] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48.

[32] M. J. L. de Jong, "Classifying human pilot skill level using deep artificial neural networks," MSc. thesis, Delft University of Technology, Faculty of Aerospace Engineering, 2021, unpublished. [Online]. Available: http://repository.tudelft.nl

[33] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, p. 249–259, Oct 2018.

[34] J. Johnson and T. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, pp. 1–54, 03 2019.

[35] C. S. Draper, H. Whitaker, and L. R. Young, "The roles of men and instruments in control and guidance systems for spacecraft," in *15th International Astronautical Congress, Poland*, 1964.

[36] A. E. Preyss and J. L. Meiry, "Stochastic modeling of human learning behavior," *IEEE Transactions on Man-Machine Systems*, vol. 9, no. 2, pp. 36–46, 1968.

[37] L. R. Young, "On adaptive manual control," *IEEE Transactions on Man-Machine Systems*, vol. 10, no. 4, pp. 292–331, 1969.

[38] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 2016.

[39] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data Augmentation for Time Series Classification using Convolutional Neural Networks," in *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, Riva Del Garda, Italy, Sep 2016.

[40] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, pp. 606–660, 05 2017.

[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 05 2015.

[42] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1578–1585.

[43] F. Chollet *et al.*, "Keras," 2015, software available from keras.io.

[44] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[45] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[46] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[47] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," 07 2017, arXiv: 1707.06799.

[48] M. Hermans and B. Schrauwen, "Training and analyzing deep recurrent neural networks," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. Curran Associates Inc., 2013, pp. 190—198.

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[50] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[51] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[52] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," ser. ICML'10. Madison, WI, USA: Omnipress, 2010, pp. 807–814.

[53] M. Lin, Q. Chen, and S. Yan, "Network in network," 2014, arXiv: 1312.4400.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, arXiv: 1512.03385.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, arXiv: 1409.1556.

[56] I. H. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "InceptionTime: Finding AlexNet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936—-1962, Sep 2020.

[57] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[58] J. Patterson and A. Gibson, *Deep Learning: A Practioner's Approach*, 1st ed. 1005 Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2017.

[59] D. Campbell, R. A. Dunne, and N. A. Campbell, "On the pairing of the Softmax activation and cross-entropy penalty functions and the derivation of the Softmax activation function," in *Proceedings of 8th Australian Conference on Neural Networks, Melbourne*, 1997, pp.

181–185.

[60] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[61] D. Martens, J. Vanthienen, W. Verbeke, and B. Baesens, "Performance of classification models from a user perspective," *Decision Support Systems*, vol. 51, no. 4, pp. 782–793, 2011, recent Advances in Data, Text, and Media Mining Information Issues in Supply Chain and in Service System Design.

[62] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 210–215.

[63] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2921–2929.

[64] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, no. 2, p. 336–359, Oct 2019. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7

[65] S. Lipovetsky and M. Conklin, "Analysis of regression in game theory approach," *Applied Stochastic Models in Business and Industry*, vol. 17, no. 4, pp. 319–330, 2001.

[66] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and information systems*, vol. 41, no. 3, 2014.

[67] A. Datta, S. Sen, and Y. Zick, "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 598–617.

[68] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1135—-1144.

[69] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 3145–3153.

[70] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *Technical Report, Univeristé de Montréal*, 01 2009.

[71] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2528–2535.

[72] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5188–5196.

[73] L. S. Shapley, *A Value for n-Person Games*. Princeton University Press, 2016, vol. 2, pp. 307–318.

[74] C. L. X. C. Zhuwei Qin, Fuxun Yu, "How convolutional neural networks see the world — a survey of convolutional neural network visualization methods," *Mathematical Foundations of Computing*, vol. 1, no. 2, pp. 149–180, 2018.

[75] E. S. Krendel and D. T. McRuer, "A servomechanisms approach to skill development," *Journal of the Franklin Institute*, vol. 269, no. 1, pp. 24–42, 1960.

[76] P. Hall, "On the art and science of machine learning explanations," 2020, arXiv: 1810.02909.

[77] D. T. McRuer and H. R. Jex, "A review of quasi-linear pilot models," *IEEE Transactions on Human Factors in Electronics*, vol. HFE-8, No. 3, pp. 231–249, 1967.

[78] D. T. McRuer, D. Graham, E. S. Krendel, and R. J. W., *Human Pilot Dynamics in Compensatory Systems: Theory, Models, and Experiments with Controlled Element and Forcing Function Variations*. Air Force Flight Dynamics Laboratory, Wrigth-Patterson AFB (OH), August 1965.

[79] W. H. Levison, S. Baron, and D. L. Kleinman, "A model for human controller remnant," *IEEE Transactions on Man-Machine Systems*, vol. 10, no. 4, pp. 101–108, 1969.

[80] K. van der El, D. M. Pool, and M. Mulder, "Analysis of human remnant in pursuit and preview tracking tasks," *IFAC-PapersOnLine*, vol. 52, no. 19, pp. 145–150, 2019, 14th IFAC Symposium on Analysis, Design, and Evaluation of Human Machine Systems HMS 2019.

[81] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep learning is robust to massive label noise," 2018.

[82] J. R. Norris, *Markov Chains*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[83] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[84] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 3395–3403.

[85] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[86] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015,

pp. 1520–1528.

[87] A. Jansson, E. J. Humphrey, N. Montecchio, R. M. Bittner, A. Kumar, and T. Weyde, "Singing Voice Separation with Deep U-Net Convolutional Networks." in *Proceedings of the 18th International Society for Music Information Retrieval Conference*. Suzhou, China: ISMIR, Oct. 2017, pp. 745–751.

[88] K. Fauvel, T. Lin, V. Masson, Élisa Fromont, and A. Termier, "XCM: An explainable convolutional neural network for multivariate time series classification," 2020, arXiv: 2009.04796.

[89] R. Assaf and A. Schumann, "Explainable deep neural networks for multivariate time series predictions," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 6488–6490.

II

Appendices to Scientific Paper

A

Cybernetic Data Augmentation

This appendix will briefly explain how cybernetic pilot modeling is used in this MSc thesis project to simulate additional pilot control behavior data for classifier training. The effectiveness of this method is tested by training the deep learning classifier with this simulated data.

A.1. Method

Quasi Linear Pilot Model

The pilot simulations are done using the quasi-linear pilot model as proposed by Pool, Harder, and van Paassen (2016). Since Pool et al. used this same model to evaluate the experiment data that is used throughout this thesis, the model parameters—except the remnant parameters—of each subject are readily available for simulation. To reduce complexity, only the fixed-base group is simulated. This means that the pilot model is reduced to Figure A.1 with motion response turned off (i.e., Data-NM is simulated). The *visual response* $H_{p_v}(s)$ of the human operator model is given by Eq. (A.1). Because human operators generate lag at frequencies below the short-period mode's natural frequency of the controlled element, but exert lead equalization at higher frequencies, a quadratic lead term had to be added to Eq. (A.1) to model such equalization dynamics (Pool, Zaal, Damveld, van Paassen, Vaart, & Mulder, 2011). In total the visual response model (Eq. (A.1)) contains six model parameters: the visual gain K_v, the visual lead time constant T_{lead}, the visual lag time constant T_{lag}, the visual time delay τ_v, and the neuromuscular dynamics modeled as a second-order mass-spring-damper (McRuer, Graham, Krendel, & W., 1965) with neuromuscular frequency ω_{nm} and neuromuscular damping ratio ζ_{nm}.

$$H_{p_v}(s) = \underbrace{K_v \frac{(T_{lead}s+1)^2}{T_{lag}s+1}}_{\text{pilot equalization}} \underbrace{e^{-s\tau_v}}_{\text{response delay}} \underbrace{H_{nm}(s)}_{\text{actuation dynamics}} \tag{A.1}$$

$$H_{nm}(s) = \frac{\omega_{nm}^2}{s^2 + 2\zeta_{nm}\omega_{nm}s + \omega_{nm}^2} \tag{A.2}$$

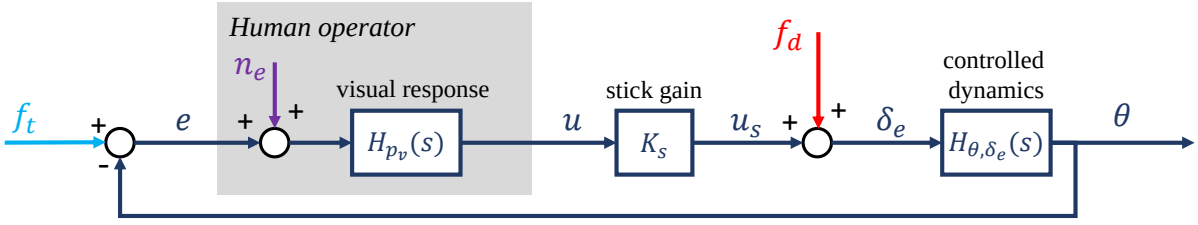


Figure A.1: Simplified quasi linear pilot model with only visual response. The remnant behavior is modeled as colored noise n_e injected at the error.

Remnant Parameters Estimation

As previously mentioned, Pool et al. (2016) did not evaluate the remnant of the experiment participants. Therefore, a method must be introduced to evaluate—and ultimately *simulate*—the remnant. The remnant n can be modeled as filtered white noise (colored noise) injected at the error, i.e. Eq. (A.3). As proposed by Levison, Baron, and Kleinman (1969), a first-order low-pass filter (Eq. (A.4)) can be used to generate the colored noise. Three steps are taken to obtain the filter parameters K_n and $T_{n,lag}$:

$$S_{nn_e}(j\omega) = |H_n(j\omega)|^2 S_{ww}(j\omega) \tag{A.3}$$

$$H_n(j\omega) = K_n \frac{1}{1 + T_{n,lag} j\omega} \tag{A.4}$$

Step 1) Estimate S_{uu_n}: This step is based on the premise that the control output power-spectral density function S_{uu} is the sum of the output spectra due to target, disturbance, and remnant (i.e., Eq. (A.5)). This is under the assumption that these signals are linearly independent (Levison et al., 1969). The spectrum $S_{uu}(j\omega)$ can be estimated from discrete time measurements using Eq. (A.6) (van der El, Pool, & Mulder, 2019). Here L is the number of recorded time steps, f_s the sampling frequency in Hz, and $U(j\omega)$ is the Discrete Fourier Transform of the control output $u(t)$. To decrease noise in the power-spectral density function, S_{uu} was calculated as the average S_{uu} of five consecutive runs. Once S_{uu} is computed, S_{uu_d} and S_{uu_t} can be identified as the power-spectral density at the target and disturbance frequencies, respectively. The power spectral density of the control input due to remnant S_{uu_n} can be recognized as the power-spectral density outside of the forcing function frequencies. Interpolation is used to estimate S_{uu_n} at $\omega_{d,t}$. An example of the above is depicted in Figure A.2.

$$S_{uu}(j\omega) = S_{uu_t}(j\omega) + S_{uu_d}(j\omega) + S_{uu_n}(j\omega) \tag{A.5}$$

$$S_{uu}(j\omega) = \frac{1}{f_s L} \left| U(j\omega) \right|^2 \tag{A.6}$$

Step 2) Calculate S_{nn_e}: In compensatory tasks, the power-spectral density of the remnant injected at the error can be estimated at the target and disturbance frequencies using the following equations (van der El et al., 2019):

$$\begin{aligned} S_{nn_e}(j\omega_d) &= \frac{S_{uu_n}(j\omega_d)}{S_{uu_d}(j\omega_d)} S_{f_d f_d}(j\omega_d) \\ S_{nn_e}(j\omega_t) &= \frac{S_{uu_n}(j\omega_t)}{S_{uu_t}(j\omega_t)} S_{f_t f_t}(j\omega_t) \end{aligned} \tag{A.7}$$

Step 3) Approximate low-pass filter parameters: In the third and final step, the model parameters of the low-pass filter to produce colored noise are determined. This is done by first rewriting $|H_n|^2$ in Eq. (A.3) to Eq. (A.8), and then applying a least squares cost function to fit Eq. (A.8) to the estimated remnant injected at the error spectral density S_{nn_e}. An example of this approach is shown in Figure A.3. Since S_{uu} was averaged over five runs, each subject will have a constant remnant setting per five runs.

$$\begin{aligned} |H_n| &= K_n \left| \frac{1}{1 + T_{n,lag} j\omega} \right| = K_n \frac{1}{\sqrt{1 + T_{n,lag}^2 \omega^2}} \\ \rightarrow |H_n|^2 &= K_n^2 \frac{1}{1 + T_{n,lag}^2 \omega^2} \end{aligned} \tag{A.8}$$

Testing Augmentation Effectiveness

The effectiveness of the proposed data augmentation method is tested by *training* the deep learning classifier with the *simulated* pilot data and then *validating* the trained classifier on *real* pilot data. The optimized hyperparameters as were presented in the research paper are used for this analysis.

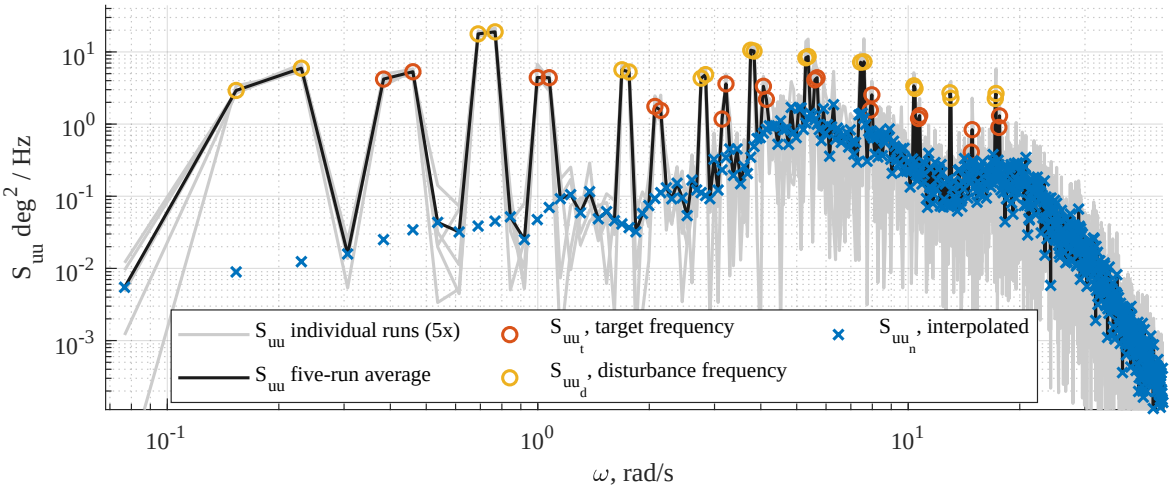


Figure A.2: Example of control output spectrum of a single participant of Data-NM. Interpolation is used to estimate S_{uu_n} at target- and disturbance frequencies.

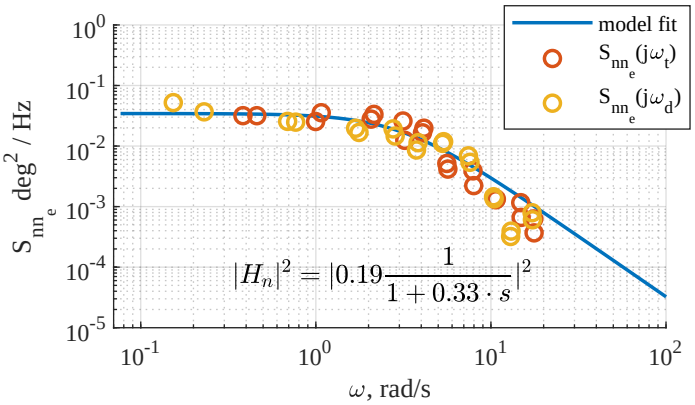


Figure A.3: Example of fitting low pass filter remnant model to estimated remnant (injected at the error) at target and disturbance frequencies. Data of a single participant from Data-NM.

A.2. Results

The remnant parameters found using the described method are shown in Figure A.4, here it can be seen that the low-pass filter gain K_n decreases as the participants gain more experience, this means that the amplitude of the remnant reduces with training. Similarly, the low-pass filter lag time-constant $T_{n,lag}$ declines too, this increases the break frequency ($\omega_b = 1/T_{n,lag}$) of the remnant signal, meaning that remnant becomes more white/less colored after training.

Simulating Pilot Control Behavior

A complete overview of the pilot model parameters that are used to simulate 'unskilled' (at first 20 runs) and 'skilled' (at last 20 runs) pilot behavior are shown in Figure A.5. To validate if the simulated control behavior accurately represents the true pilot control behavior, the variance of the tracking error $\sigma^2(e)$ and the control input $\sigma^2(u)$ of real- and simulated tracking runs are compared. This comparison is shown in Figure A.6, here each tracking run and its simulated counterpart are displayed by circles and crosses, respectively. Notice how on average, the trend of the simulated data is similar to the trend of the real pilot data. As an example, Figure A.7 shows time traces of actual pilot control behavior next to its simulated counterpart. Although the simulated- and real tracking data are similar, there are noticeable discrepancies between the two. This is likely due to two effects: 1) the linear response of the human operator is time variant, whereas the simulation model assumes the linear part of the control behavior to be a constant transfer function model over the entire tracking run. 2) the nonlinear response of the human operator (remnant) is modeled as colored noise, i.e. the time domain realization of this signal is stochastic and is thus not expected to precisely track the nonlinear portion of the time traces of the human operator. To show the effect of adding remnant to the pilot simulation model, a simulated time trace *without* remnant is also depicted. It can be seen that the simulated behavior *with* remnant more accurately mimics the real control behavior, both showing high frequency noisy behavior on top of the smooth linear response.

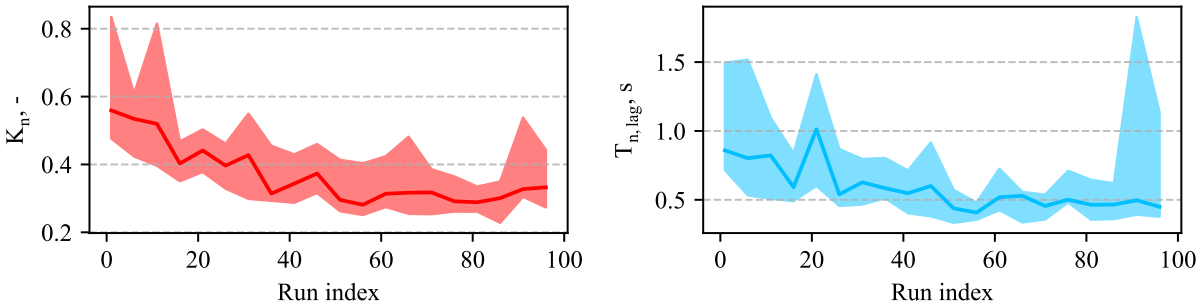


Figure A.4: Results of remnant describing low pass filter parameters for all *fixed base* participants. *Data set = Data-NM*

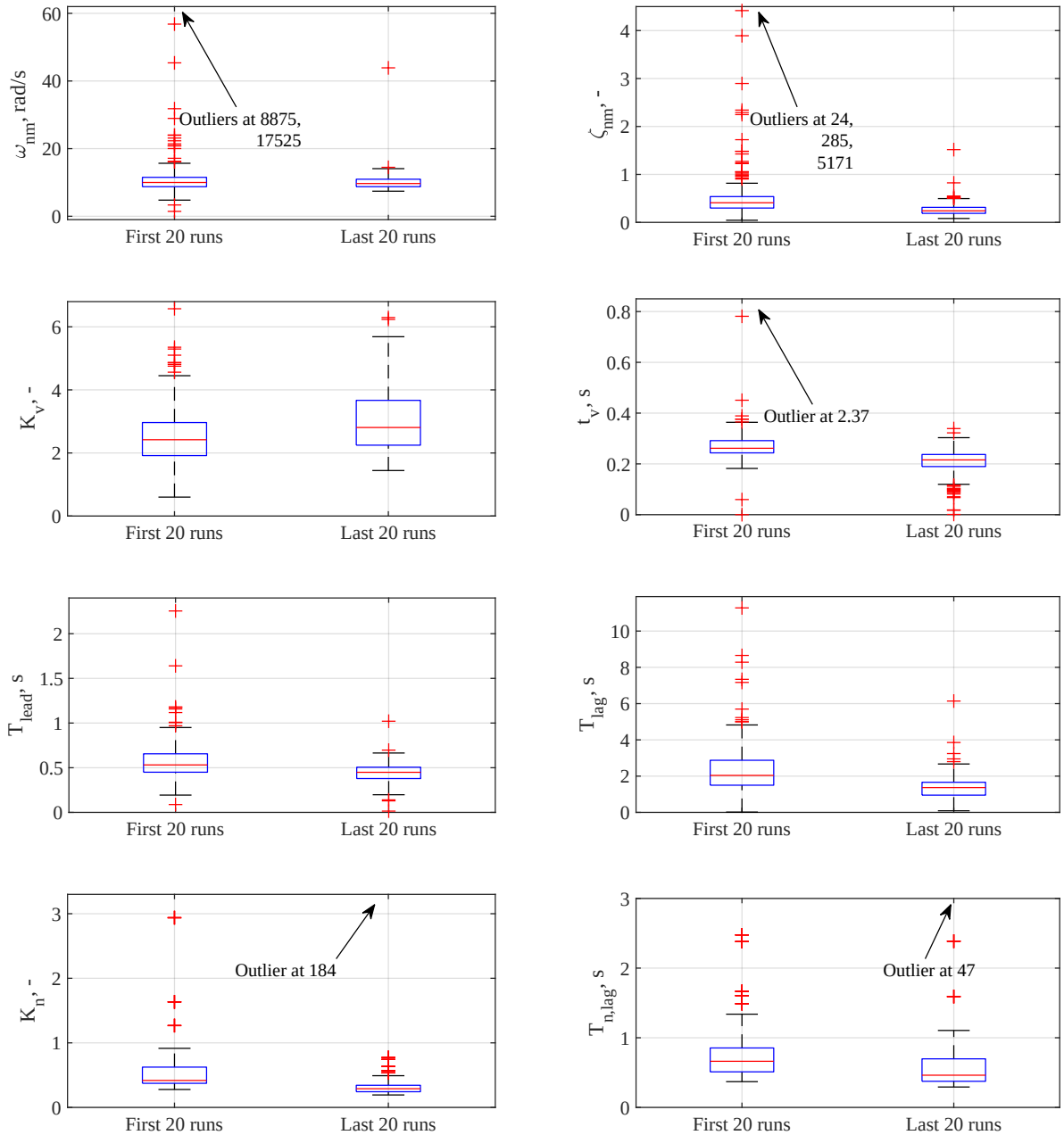


Figure A.5: Overview of pilot describing parameters used to simulate *fixed base* 'skilled' (last 20 runs) and 'unskilled' (first 20 runs) pilot behavior. *Data set = Data-NM*

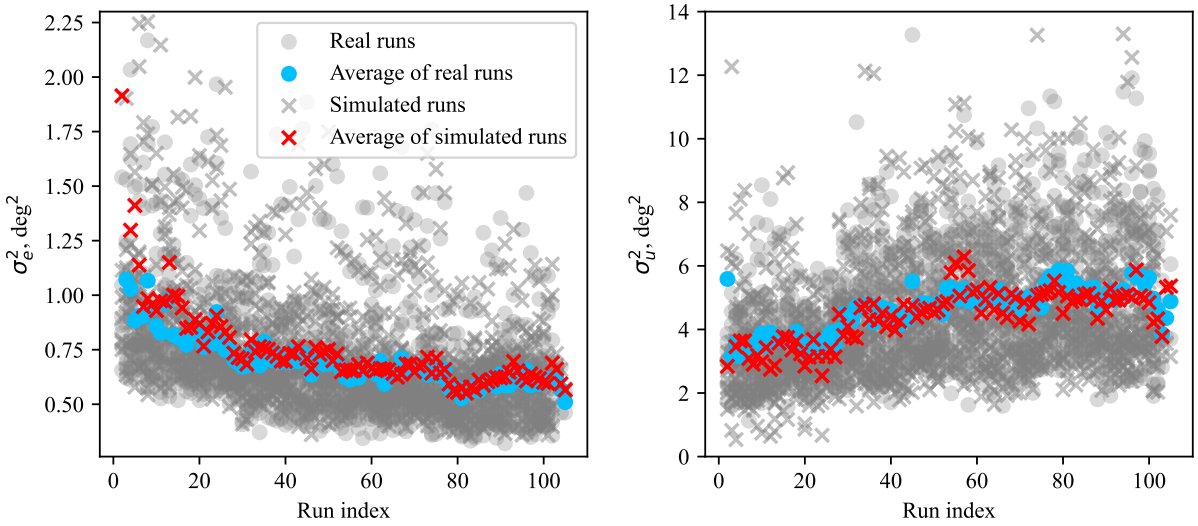


Figure A.6: Comparison of variance in e and u for simulated pilot behavior and actual tracking runs. *Data set = Data-NM*

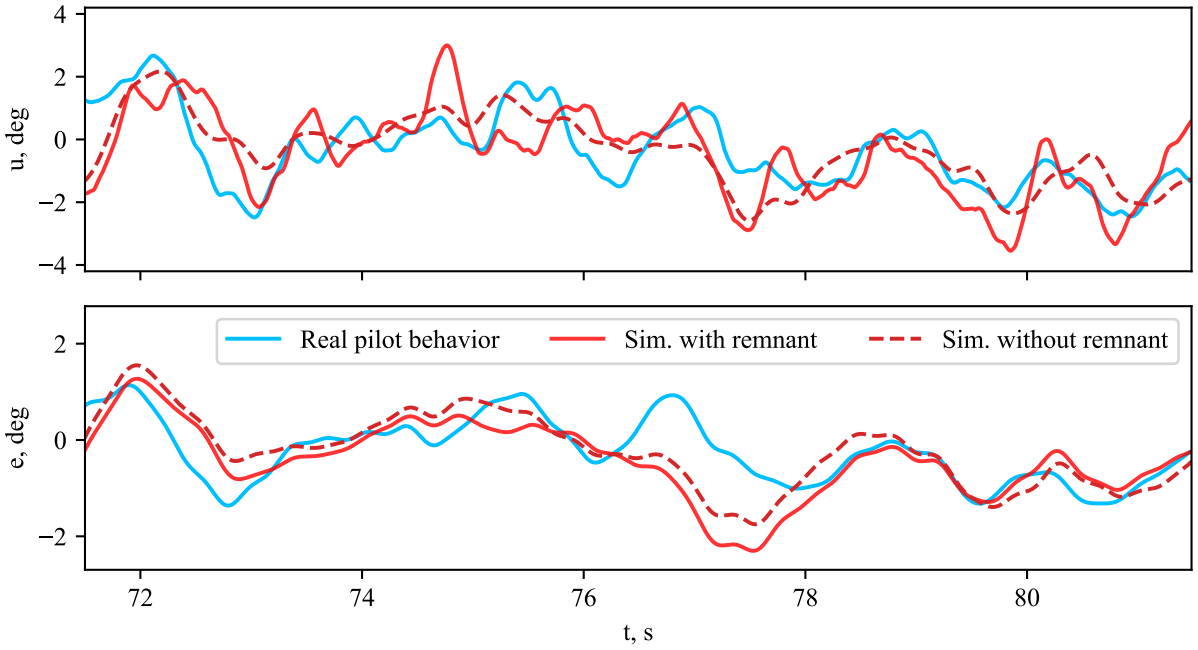


Figure A.7: Example of time trace of simulated behavior with/without remnant versus actual pilot control behavior. *Data set = Data-NM*

Testing Augmentation Effectiveness

The effectiveness of the described cybernetic data augmentation method is empirically tested. This is done by training the optimized deep learning model with 30 randomly split distributions of 80% train data and 20% validation data. Here the 80% train data is either 'simulated pilot data *with* remnant', 'simulated pilot data *without* remnant', or 'real pilot data' (for comparison). The 20% validation data is *always* real pilot data. To get a fair measure of effectiveness, it is ensured that the real counterparts of simulated runs used for training are *not* present in the validation set (e.g., if the *simulated* version of tracking run number 85 by subject 3 is used to train the classifier, then the *real* tracking run 85 by subject 3 can *not* be used as validation data). The results of this analysis are shown in Figure A.8, here each dot represents one of the 30 random 80%/20% train/validation splits. The y-axis shows the validation accuracy when training with the respective data sets and validating on real pilot data. Additionally, the x-axis indicates the bias of the trained model (expressed as the percentage of samples classified as 'skilled'). Ideally, the bias is 50%; since the underlying validation data is evenly distributed between each class, 50% 'skilled' predictions means the model is unbiased.

Evidently, the addition of remnant to the pilot model increases the validation accuracy, confirming that the trained classifier incorporates the nonlinear portion of control behavior in its predictions. However, the model trained with simulation data *including* remnant has an overall bias towards 'skilled' predictions. It can be hypothesized that this induced bias indicates an overestimation of remnant in the pilot simulations. Namely, through the simulated data the model is taught that 'skilled' behavior has less remnant than 'unskilled' behavior (i.e., K_n is lower in 'skilled' simulations than in 'unskilled' simulations, as was shown in Figure A.5). Having learned this pattern during training, the model is now validated on real pilot behavior, if the overall remnant in the real pilot control behavior (validation data) is lower than in the simulated behavior (training data), then the classifier is expected to be biased towards 'skilled' predictions.

An equal but opposite hypothesis can not be made about the observed bias towards 'unskilled' predictions for the classifiers trained with simulation data *excluding* remnant, because these classifiers are never confronted with different levels of remnant during training (i.e., $K_n = 0$ for both 'skilled' and 'unskilled' samples). This means that the source of this bias is completely accountable towards the linear portion of the control simulations. However, the observed bias is not as strong as the bias seen when training with simulated data *including* remnant (i.e., there are purple crosses left and right of the 50% line). Additionally, the spread in biases is a lot larger, indicating a bigger uncertainty in classification performance of the classifiers trained without remnant. The relation between the level of remnant in the training data and the resulting behavior of the trained classifier will have to be further analyzed in future research.

Overall, the accuracy with which the classifier—trained on simulated data—can predict the skill level of real pilot tracking behavior is still promising. However, in its current state, this cybernetic approach to data augmentation is not yet optimal. Namely, the found validation accuracy falls approximately 15%-25% short compared to validation accuracy found when training with real pilot data. This gap in accuracy—combined with the earlier discussed sensitivity towards the presence of remnant in the simulations—signify that the nonlinear part of human control behavior plays a significant role in the skill level predictions made by the classifier. The lacking validation accuracy found when training on simulation data *including* remnant, implies that the remnant model—as implemented in this research—does not fully cover the complex nonlinear behavior of real human subjects in the time domain. More research will have to be performed to test the effectiveness of this cybernetic data augmentation method, perhaps it is beneficial in circumstances where there is only very little training data available. Additionally, this 'deep learning review' of the remnant model should be extended, so that the shortcomings of the current remnant model can be targeted and improved.

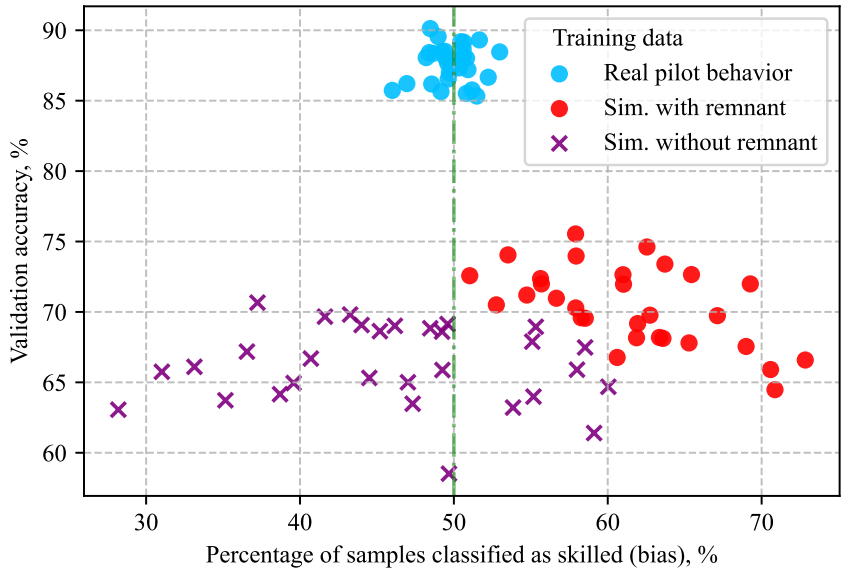


Figure A.8: Testing the effectiveness of cybernetic data augmentation by training the model on simulated data—with- and without remnant—and validating it on real pilot data. Each dot indicates a random 80%/20% distribution of train/validation data. Introduction of remnant to the pilot simulations induces bias in the trained classifier's predictions. Validation data = Data-NM

B

Additional Optimization Results

This appendix will present additional results that were found during optimization of the data settings and model hyperparameters. The main findings of the classifier optimization have been discussed in the scientific paper.

B.1. Random Seed vs Performance

The contents of this particular section are not necessarily additional optimization results, but more so an important observation that was made during optimization. Namely, when testing the performance of different artificial neural network architectures, it was discovered that specific (random) distributions of (80%) train and (20%) validation data consistently led to better validation accuracy than others. This behavior is captured in Figure B.1, where the random seeds that determine the train/validation split are ranked by the resulting average performance across the four tested neural network architectures. Evidently, a significant trend can be observed: there is more than 6% difference between the average classification accuracy of the 'worst' and the 'best' random seed. Not only does this stress the influence of the selection of training and validation data, but more so does it emphasize the importance of testing a large set of (fixed) random seeds to obtain a fair comparison of performance.

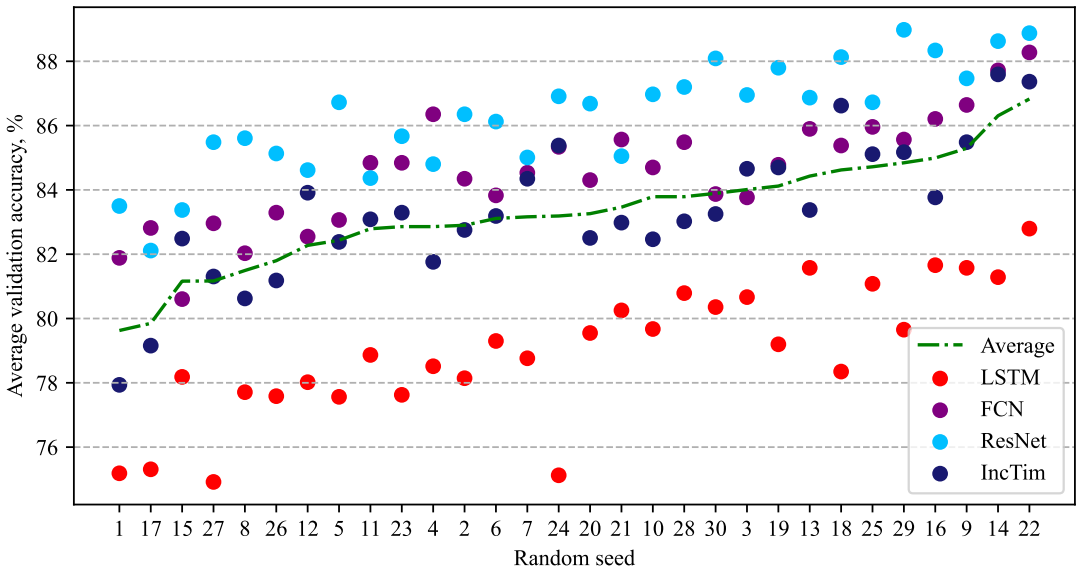


Figure B.1: Specific (random) distributions (80%/20%) of the training and validation data result in better validation accuracy across the different deep learning classifiers. Sorted by average performance across architectures, worst random seed left, best random seed right. *Data set = Data-NM*

To investigate the nature of the classifiers' preference for certain random seeds, the composition of the best performing random seed (22) is compared to the composition of the worst random

seed (1). Figure B.2 shows the three subjects that were most present (number of data points) in the train/validation set of random seed 22 and 1. As hypothesized in the scientific article, the difference in validation accuracy is more likely caused by the composition of the *validation* set than by the composition of the *training* set. This hypothesis is substantiated by evidence that deep learning classifiers are resilient towards label noise in the train set (Rolnick, Veit, Belongie, & Shavit, 2018). Figure B.2 shows that the worst distribution (random seed 1) has subject 7's behavior as the largest contributor to the validation data (11.7%), with 58.8% being 'skilled' samples. As presented in the scientific article, the test accuracy of the 'skilled' portion of subject 7 is merely 5.4%. This could be an explanation for the poor average validation accuracy of random seed 1. Contrarily, random seed 22 has more balanced validation data with subjects that showed better test accuracy in the scientific paper.

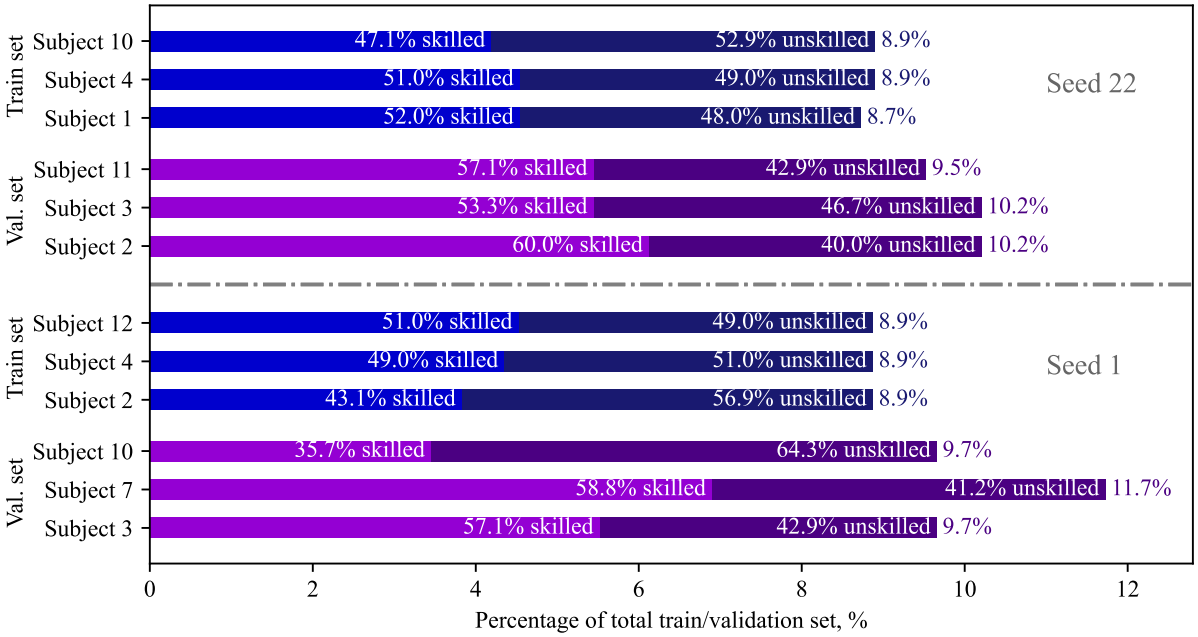


Figure B.2: The top 3 subjects with most data points in the train/validation set for the best random seed 22, and the worst random seed 1. *Data set = Data-NM*

B.2. Learning Rate

As mentioned in the scientific paper, a learning rate of 5e-4 was used throughout the optimization process. This section will briefly discuss how this learning rate was found and how it also affected the number of epochs used during training.

The range where the optimal learning rate is located can be found by using a technique described by Smith (2017). By training the network with a very low learning rate, for example 1e-6, and then increasing the learning rate exponentially after every mini-batch (training step), the loss versus learning rate can be graphed as shown in Figure B.3. The range where the decrease in loss is the steepest indicates the range of optimal learning rate values. Smith (2017) uses this range to set bounds for a cyclical learning rate (i.e., going up and down during training) as this was found to improve classification accuracy (Smith, 2017). In this current research a constant learning rate was used and the range shown in Figure B.3 was used as an indication of learning rate values to test during optimization.

The learning rate values that were tested are 1e-4, 5e-4, 1e-3, 5e-3, and 1e-2. The results of this analysis are shown in Figure B.4. Based on these findings, the optimal learning rate was determined to be 5e-4, as this resulted in the highest validation accuracy on average.

It was also tested how these different learning rates affected the best epoch (i.e., epoch with minimum validation loss), this is shown in Figure B.5. Here it can be observed how both too large and too small learning rate increase training time. As is explained in Figure 3.4, too small learning rates mean the model takes very small update steps, whereas too large learning rates makes the model overshoot optima, both increasing training time. Based on Figure B.5, it was decided that—at 5e-4—a total of 20 epochs would be sufficient to train the classifier.

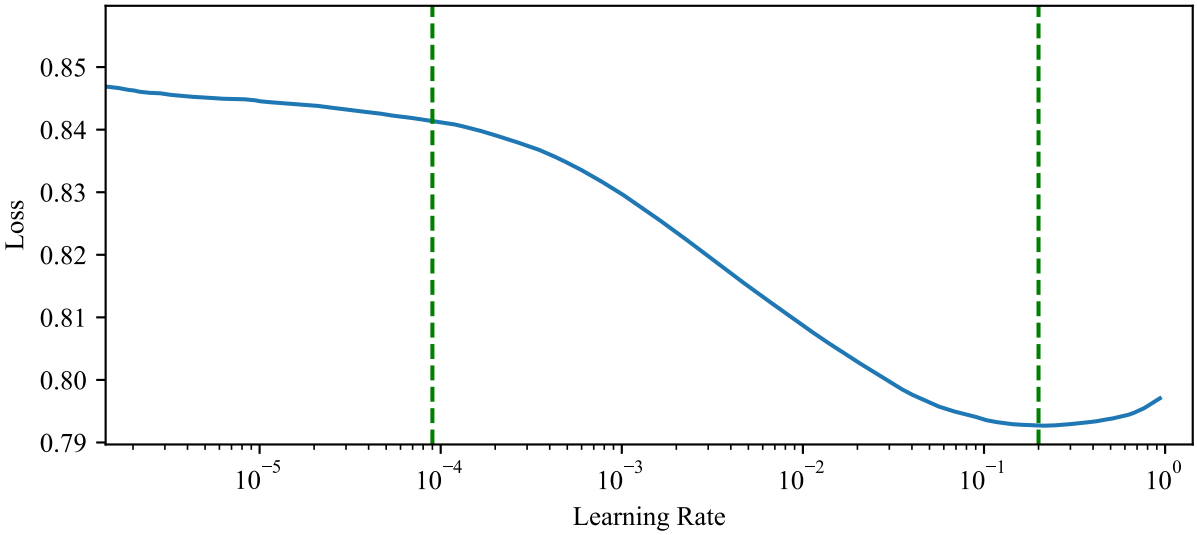


Figure B.3: Increments of the learning rate after each training step, reveal the optimal learning rate range. *Data set = Data-NM*

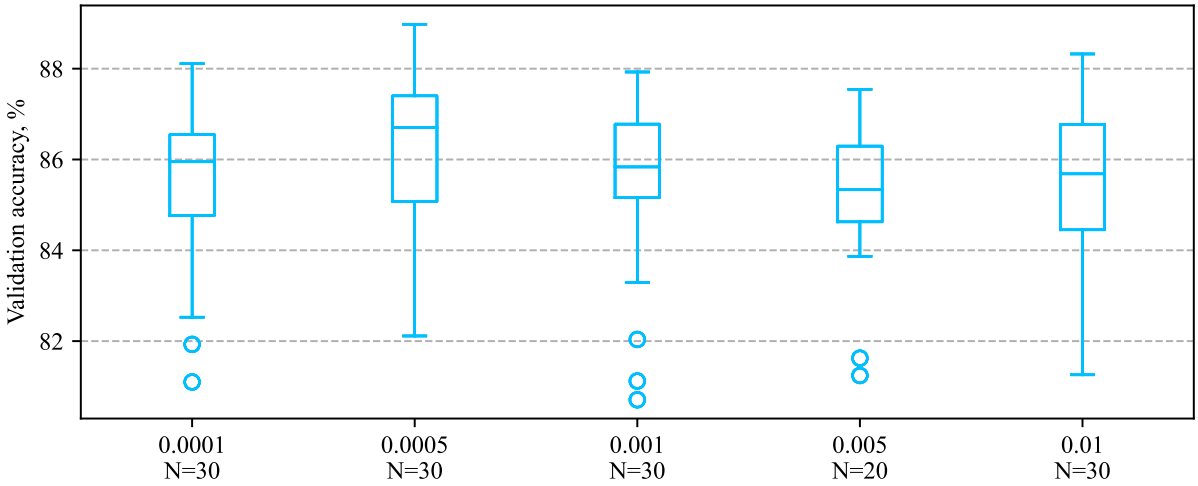


Figure B.4: Box plots of found validation accuracy when training with different learning rates. *Data set = Data-NM*

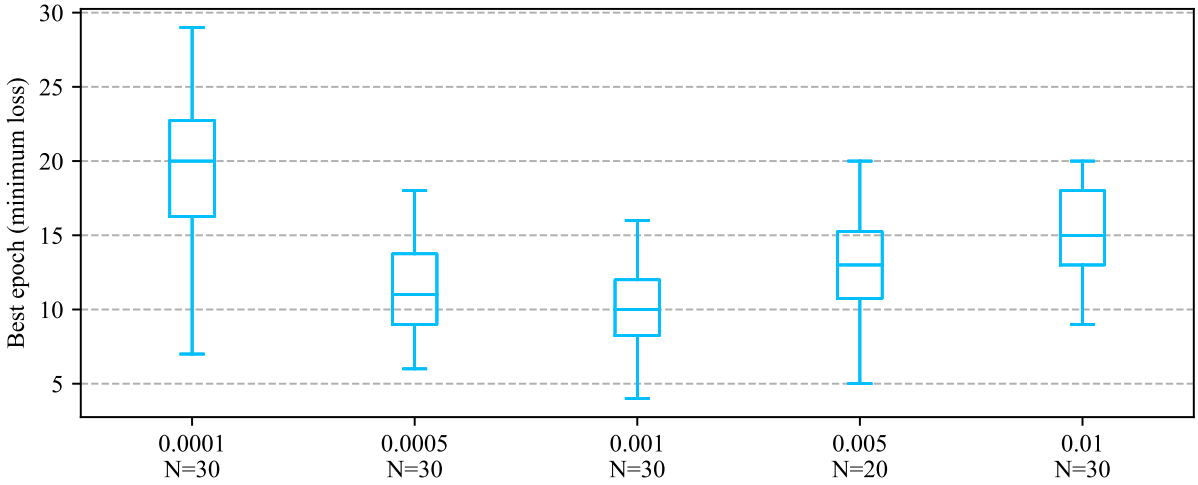


Figure B.5: Box plots of best epoch when training with different learning rates. *Data set = Data-NM*

B.3. Variable Selection

Table B.1 shows all results of the variable selection optimization analysis (the scientific paper only displayed a summarized table). Additionally, Table B.2 shows the results of the input optimization for the 20 s window size sampled at 25 Hz. It can be observed how the top- and bottom 10 combinations are similar between the two window settings. The order in which the single-variable-inputs are ranked is nearly identical: $\dot{u}$, u, e, x, $\dot{x}/\dot{e}$.

To make comparison of the two tables easier, certain input combinations are color coded based on their rank in the $WS = 1.6$ s, $S_f = 50$ Hz setting (left table): rank 1-5 are colored **blue**, rank 6-10 are colored **violet**, rank 54-58 are colored **purple**, and rank 59-63 are colored **red**.

B.4. Dimension of Convolutional Layers

As was mentioned in the scientific paper, a comparison was made between using 1D convolutional layers and 2D convolutional layers in the ResNet architecture (with all other settings kept at their base values). The results of this comparison are shown in Figure B.6. Evidently, 1D convolutional layers result in better classification performance than 2D layers. Using 2D convolutional layers also resulted in 2,859,074 trainable parameters instead of 505,986 trainable parameters of the 1D architecture under nominal settings. The increased number of parameters resulted in 5 times longer training times.

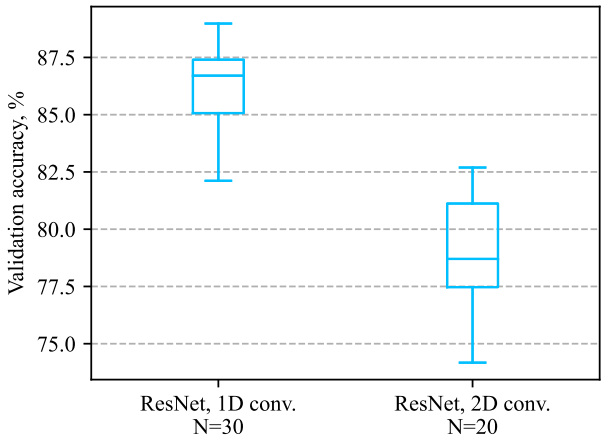


Figure B.6: Comparison of validation accuracy found when using 1D convolutional layers or 2D convolutional layers in the proposed ResNet architecture. *Data set = Data-NM*

Table B.1: Ranked performance of input combinations when using window size of **1.6 s** sampled at **50 Hz**.
Data set = Data-NM

Rank	Input combination	Val. acc.	Val. loss
1	$u + \dot{u} + e + \dot{e}$	86.35 %	0.327
2	$u + \dot{u} + e + \dot{e} + \dot{x}$	84.96 %	0.3417
3	$u + \dot{u} + e + \dot{e} + x$	84.71 %	0.3486
4	$u + \dot{u} + e + \dot{e} + x + \dot{x}$	83.68 %	0.3662
5	$u + \dot{u} + x + \dot{x}$	82.82 %	0.3876
6	$u + \dot{u}$	82.82 %	0.3846
7	$\dot{u} + e + \dot{e}$	82.68 %	0.3796
8	$u + \dot{u} + e$	82.42 %	0.3984
9	$u + \dot{u} + e + x + \dot{x}$	82.04 %	0.4123
10	$u + \dot{u} + \dot{e} + x + \dot{x}$	81.94 %	0.4025
11	$\dot{u} + e + \dot{e} + \dot{x}$	81.85 %	0.3998
12	$u + \dot{u} + x$	81.83 %	0.4037
13	$\dot{u} + e + \dot{e} + x$	81.43 %	0.4006
14	$u + \dot{u} + e + x$	81.29 %	0.4185
15	$u + \dot{u} + \dot{x}$	81.2 %	0.4157
16	$u + \dot{u} + \dot{e}$	81.08 %	0.4183
17	$u + \dot{u} + e + \dot{x}$	80.51 %	0.4194
18	$u + \dot{u} + \dot{e} + x$	80.4 %	0.4264
19	$u + \dot{u} + \dot{e} + \dot{x}$	80.29 %	0.432
20	$\dot{u} + e + \dot{e} + x + \dot{x}$	80.09 %	0.4156
21	$\dot{u} + e + x + \dot{x}$	79.33 %	0.4467
22	$u + e + \dot{e}$	79.26 %	0.4249
23	$\dot{u} + x + \dot{x}$	78.96 %	0.4412
24	$u + e + \dot{e} + \dot{x}$	78.67 %	0.4343
25	$u + e + \dot{e} + x$	78.42 %	0.4293
26	$\dot{u} + e$	78.38 %	0.472
27	$u + e + \dot{e} + x + \dot{x}$	77.73 %	0.4453
28	$\dot{u} + e + \dot{x}$	77.3 %	0.48
29	$\dot{u} + e + x$	77.19 %	0.4872
30	$\dot{u} + \dot{e} + x + \dot{x}$	77.18 %	0.4644
31	$\dot{u}$	76.92 %	0.488
32	$u + e + x + \dot{x}$	76.57 %	0.4776
33	$u + x + \dot{x}$	76.4 %	0.4762
34	$u + \dot{e} + x + \dot{x}$	76.2 %	0.4855
35	$\dot{u} + x$	75.85 %	0.5045
36	$\dot{u} + \dot{e}$	75.12 %	0.513
37	$\dot{u} + \dot{e} + x$	75.08 %	0.5149
38	$u + e$	74.46 %	0.5203
39	$u + e + \dot{x}$	74.43 %	0.5151
40	$e + \dot{e}$	74.31 %	0.4774
41	$\dot{u} + \dot{e} + \dot{x}$	74.2 %	0.5305
42	$e + \dot{e} + x$	74.19 %	0.4794
43	$\dot{u} + \dot{x}$	73.8 %	0.5234
44	$e + \dot{e} + \dot{x}$	73.71 %	0.4816
45	$u + e + x$	73.63 %	0.5335
46	$e + \dot{e} + x + \dot{x}$	73.58 %	0.4859
47	$\dot{e} + x + \dot{x}$	73.38 %	0.5247
48	$x + \dot{x}$	73.34 %	0.5197
49	$e + x + \dot{x}$	73.26 %	0.516
50	u	73.15 %	0.535
51	$u + \dot{e} + x$	72.4 %	0.5455
52	$u + x$	72.29 %	0.5483
53	$u + \dot{e}$	70.98 %	0.5639
54	$u + \dot{x}$	70.65 %	0.5709
55	$e + \dot{x}$	70.36 %	0.5621
56	$u + \dot{e} + \dot{x}$	70.09 %	0.5805
57	$e + x$	68.78 %	0.5811
58	$\dot{e} + x$	68.56 %	0.5929
59	e	65.61 %	0.6108
60	x	63.58 %	0.6391
61	$\dot{x}$	63.01 %	0.6382
62	$\dot{e}$	61.71 %	0.6478
63	$\dot{e} + \dot{x}$	58.96 %	0.6729

Table B.2: Ranked performance of input combinations when using window size of **20 s** sampled at **25 Hz**.
Data set = Data-NM

Rank	Input combination	Val. acc.	Val. loss
1	$u + \dot{u} + e + \dot{e}$	91.03 %	0.2639
2	$u + \dot{u} + e$	90.54 %	0.2489
3	$u + \dot{u} + e + \dot{x}$	90.06 %	0.2791
4	$u + \dot{u} + e + \dot{e} + x$	89.9 %	0.2758
5	$u + \dot{u} + e + x$	89.42 %	0.2658
6	$u + \dot{u} + e + \dot{e} + \dot{x}$	89.42 %	0.2962
7	$\dot{u} + e + \dot{e}$	89.1 %	0.3166
8	$u + \dot{u} + e + x + \dot{x}$	88.78 %	0.3069
9	$\dot{u} + e$	88.62 %	0.3147
10	$u + \dot{u} + e + \dot{e} + x + \dot{x}$	88.62 %	0.3039
11	$u + \dot{u} + \dot{e}$	88.46 %	0.3066
12	$u + \dot{u}$	88.3 %	0.3076
13	$\dot{u} + e + \dot{x}$	87.18 %	0.3732
14	$u + \dot{u} + \dot{x}$	86.7 %	0.3691
15	$u + \dot{u} + \dot{e} + x$	86.7 %	0.3406
16	$u + \dot{u} + \dot{e} + \dot{x}$	86.54 %	0.3581
17	$u + \dot{u} + \dot{e} + x + \dot{x}$	86.54 %	0.3554
18	$u + \dot{u} + x$	86.38 %	0.3352
19	$u + e$	86.38 %	0.3774
20	$u + \dot{u} + x + \dot{x}$	86.22 %	0.3643
21	$\dot{u} + e + \dot{e} + x$	86.06 %	0.3747
22	$\dot{u} + e + x$	85.9 %	0.3721
23	$\dot{u} + e + \dot{e} + \dot{x}$	85.9 %	0.3963
24	$\dot{u} + e + x + \dot{x}$	84.78 %	0.4194
25	$\dot{u} + e + \dot{e} + x + \dot{x}$	84.78 %	0.4124
26	$\dot{u}$	84.78 %	0.3825
27	$u + e + \dot{e}$	84.78 %	0.3645
28	$u + e + \dot{e} + \dot{x}$	84.62 %	0.4491
29	$u + e + \dot{x}$	84.29 %	0.4562
30	$\dot{u} + x$	83.97 %	0.4175
31	$u + e + x + \dot{x}$	83.97 %	0.4438
32	$u + e + x$	83.97 %	0.4818
33	$u + e + \dot{e} + x$	83.97 %	0.4385
34	u	83.97 %	0.3859
35	$u + e + \dot{e} + x + \dot{x}$	82.85 %	0.4634
36	$\dot{u} + \dot{e} + x$	82.69 %	0.4433
37	$\dot{u} + x + \dot{x}$	82.53 %	0.466
38	$\dot{u} + \dot{e}$	82.37 %	0.395
39	$\dot{u} + \dot{e} + \dot{x}$	82.21 %	0.4639
40	$\dot{u} + \dot{e} + x + \dot{x}$	82.21 %	0.4557
41	$\dot{u} + \dot{x}$	81.73 %	0.4397
42	$u + x$	81.57 %	0.4781
43	$u + \dot{e}$	81.57 %	0.4755
44	$u + \dot{e} + x$	81.09 %	0.5118
45	$u + x + \dot{x}$	79.17 %	0.5092
46	$u + \dot{x}$	79.01 %	0.541
47	$u + \dot{e} + x + \dot{x}$	78.53 %	0.5176
48	$u + \dot{e} + \dot{x}$	78.21 %	0.574
49	$e + \dot{e} + x$	75.64 %	0.5314
50	$e + \dot{e} + \dot{x}$	75.48 %	0.5752
51	$e + x$	74.52 %	0.5415
52	$e + x + \dot{x}$	74.36 %	0.5674
53	$e + \dot{e} + x + \dot{x}$	74.2 %	0.5652
54	$e + \dot{e}$	74.2 %	0.5217
55	e	74.04 %	0.5508
56	$e + \dot{x}$	73.72 %	0.5539
57	$x + \dot{x}$	73.4 %	0.5661
58	x	72.6 %	0.5797
59	$\dot{e} + x$	71.96 %	0.6261
60	$\dot{e} + x + \dot{x}$	70.99 %	0.6278
61	$\dot{e}$	66.51 %	0.6634
62	$\dot{x}$	66.19 %	0.6488
63	$\dot{e} + \dot{x}$	64.58 %	0.6552

B.5. Labeling Method

The scientific paper described a *experience*-based labeling method, and a *performance*-based labeling method. When keeping all other settings at their base values, the two methods result in the validation accuracy shown in Figure B.7. From this analysis it can be observed that 1) the performance-based labeling results in a higher average validation accuracy, and 2) the performance-based method has a smaller spread in validation accuracy. The latter observation is likely due to the more consistent labeling of the train/validation data (i.e., less label noise). Section 4.2.3 explains why the experience-based method was used nevertheless.

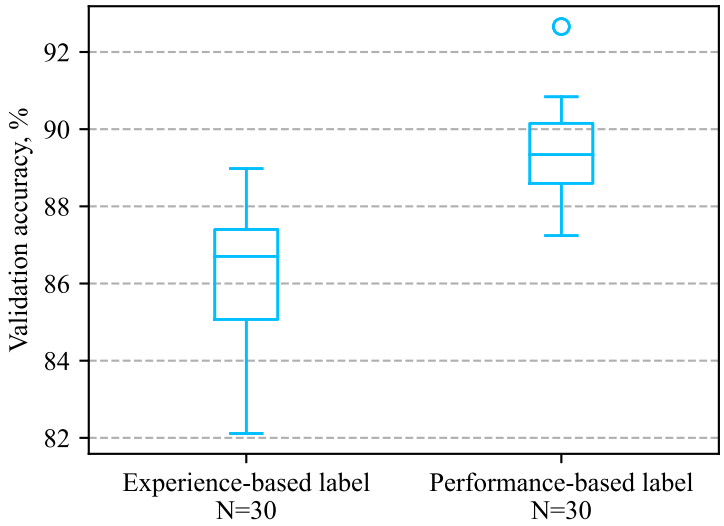


Figure B.7: Comparison of validation accuracy when using experience-based labeling (with $L_w = 15$) versus performance-based labeling to train the proposed ResNet architecture. *Data set = Data-NM*

C

Additional Performance Results

In this appendix the performance test results of the scientific paper will be extended to a different data set, window size setting, and labeling method.

Testing Performance - Motion Group

Figure C.1 shows the results of the rotating test set analysis for the moving-base group. The scientific paper already concluded that the Data-M set resulted in superior test accuracy, but this figure provides extra context by showing the classification performance of each individual subject.

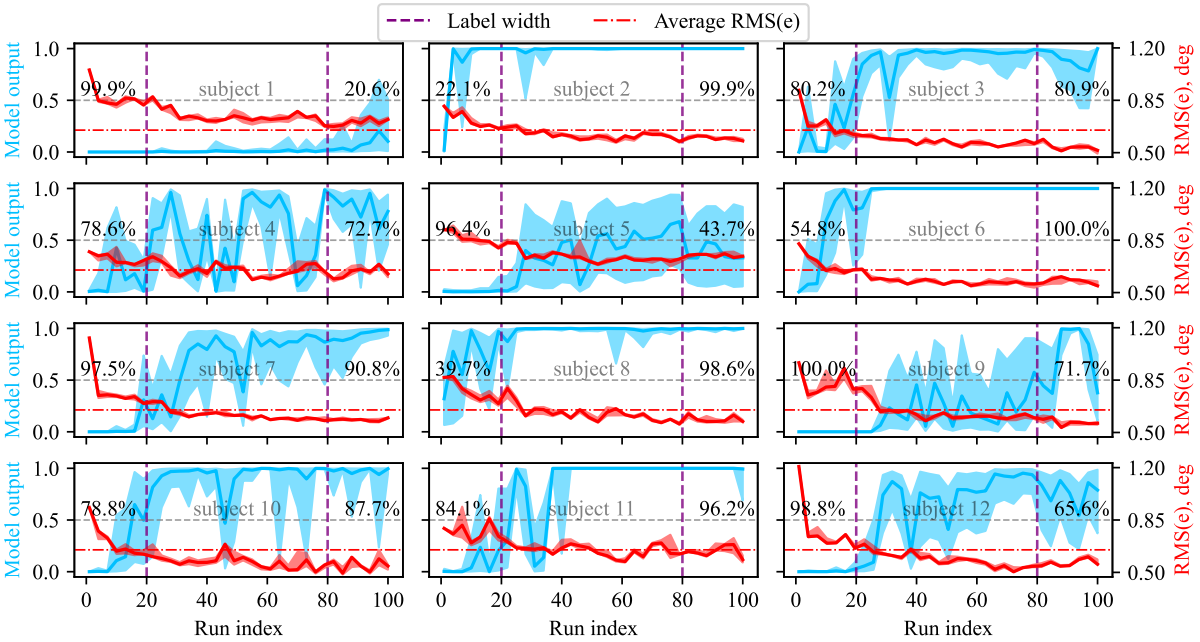


Figure C.1: Results of subject as out-of-sample test set analysis. *Data set = Data-M*

Testing Performance - No Motion Group with WS=20 s, SF=25 Hz

The window size and sampling frequency optimization indicated that a window size of 20 s sampled at 25 Hz resulted in the highest validation accuracy (but this disabled direct feedback). Figures C.2 and C.3 display the test performance of the classifier trained with these settings. Indeed the test accuracy is now higher than when using $WS = 1.2$ s and $S_f = 50$ Hz, however it can be observed that the classifier still has poor classification accuracy on subjects with an off-nominal learning curve.

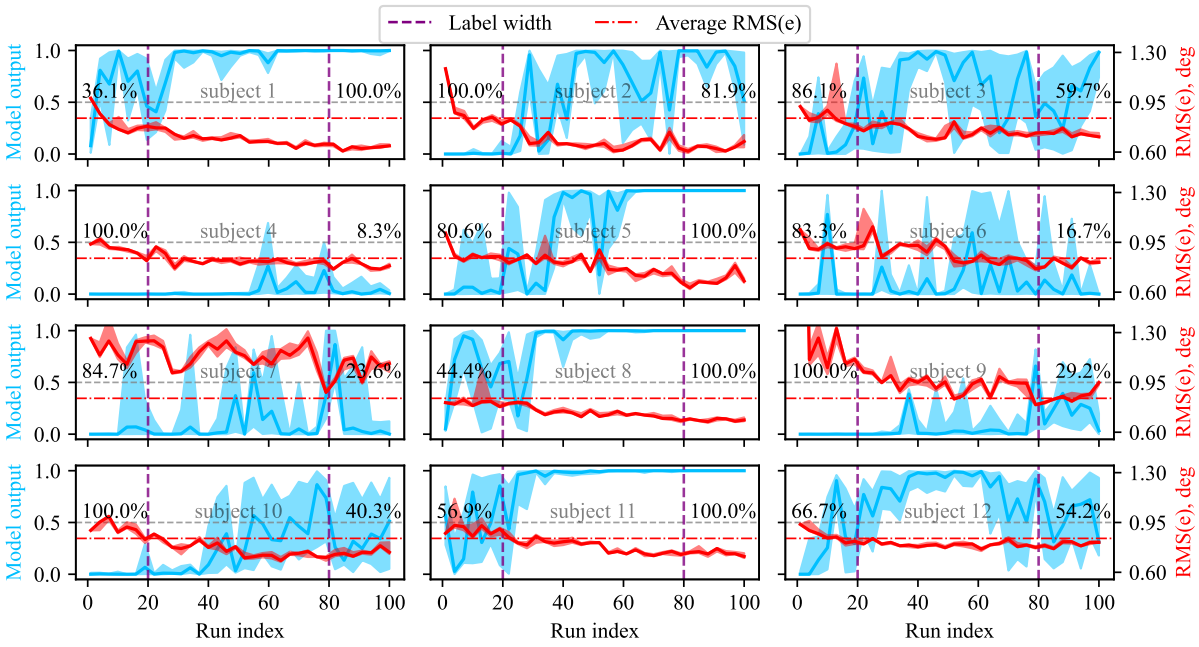


Figure C.2: Results of subject as out-of-sample test set analysis on the fixed-base group with window size of 20 seconds sampled at 25 Hz. *Data set = Data-NM*

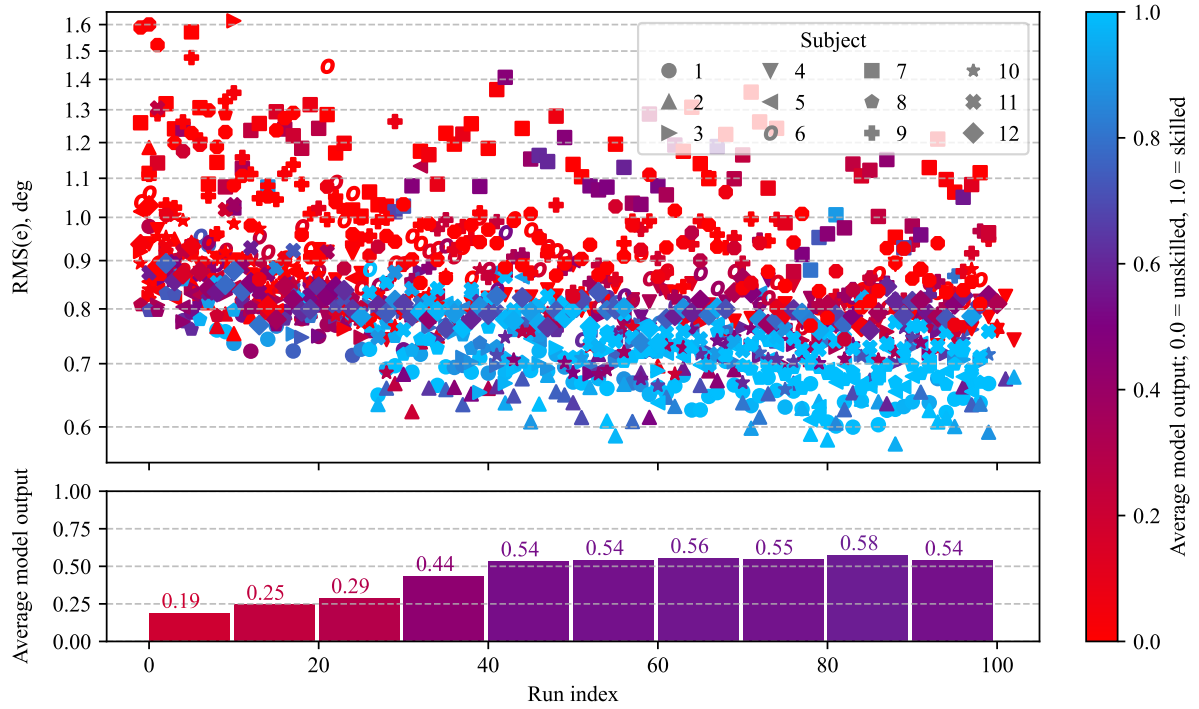


Figure C.3: Aggregated results of out-of-sample test set analysis on the fixed-base group with window size of 20 seconds sampled at 25 Hz. *Data set = Data-NM*

Testing Performance - No Motion Group with Performance-Based Labeling Method

Figures C.4 and C.5 show the test performance of the optimized classifier when using *performance*-based labeling. Figure C.4 shows that this method is particularly imprecise when subjects have an RMS(e) that is close to the average. Surprisingly, for Data-NM, the gradual increase in average model output (bottom graph Figure C.5) is more apparent for the performance-based labeling than for the experience-based labeling (Results section scientific paper).

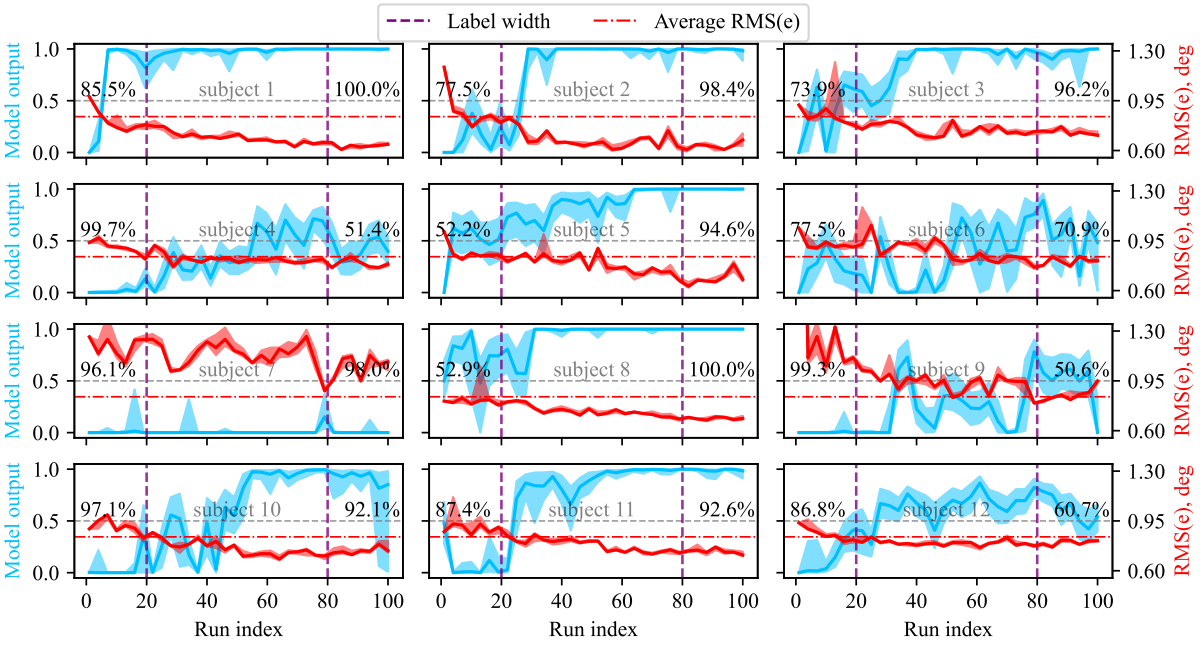


Figure C.4: Results of out-of-sample test set analysis on the fixed-base group with performance-based labeling method. *Data set = Data-NM*

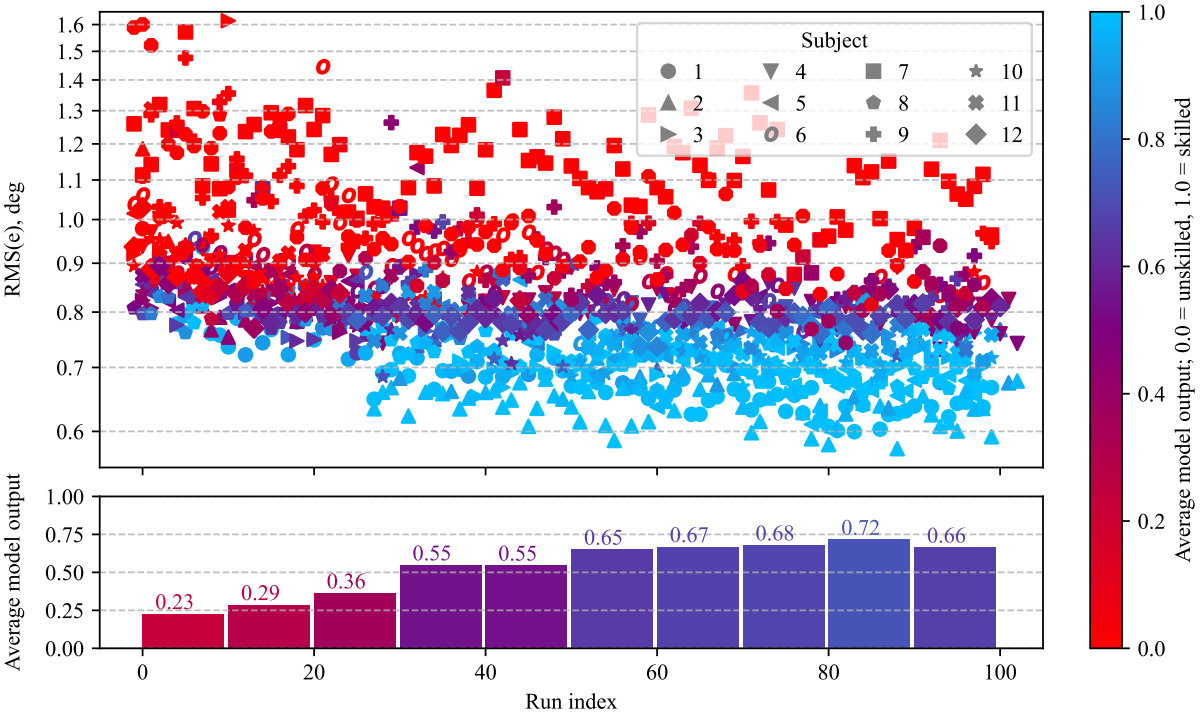


Figure C.5: Aggregated results of out-of-sample test set analysis on the fixed-base group with performance-based labeling method. *Data set = Data-NM*

D

Additional XAI Results

D.1. SHAP versus Grad-CAM

As has been explained in the scientific paper, the downside of using 1D convolutional layers is that Class Activation Mapping methods only provide 1D explanations (only temporal dynamics are preserved). Therefore, the decision was made to use SHAP as the explanation method, as this method uses an explanation model that preserves both temporal and spatial dynamics. To illustrate this trade-off, consider the sample shown in Figure D.1. This sample, belonging to the 'unskilled' class, is correctly predicted by both the 1D ResNet and 2D ResNet architectures. The explanations of these predictions—by both SHAP and Grad-CAM—are shown in Figure D.2.

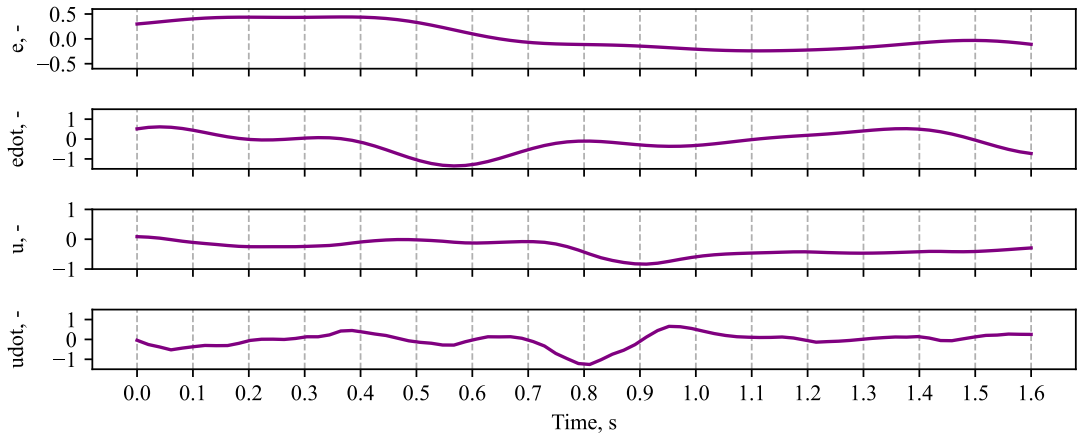


Figure D.1: Sample time traces of unskilled behavior that is explained in Figure D.2 *Data set = Data-NM*

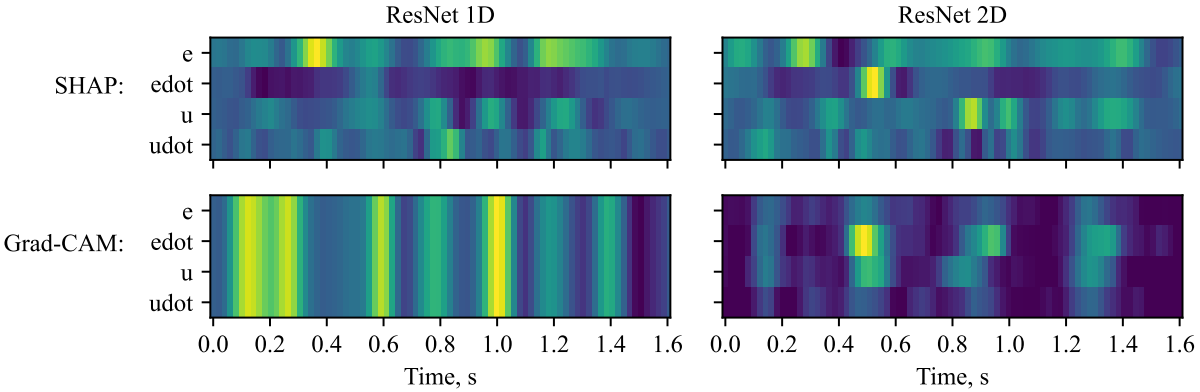


Figure D.2: Comparison of SHAP and Grad-CAM explanations of the sample shown in Figure D.1 for both the 1D and 2D ResNet architecture. The bright yellow indicates discriminative areas that led to the model output.

D.2. Qualitative SHAP Shape Comparison

In addition to the signals pertaining to skilled- or unskilled behavior shown in the scientific paper, Figures D.3, D.4, D.5, and D.6 show a qualitative comparison of signals that had a high- or low contribution towards skilled- or unskilled predictions versus the resulting model output $f(X)$. All the visualized samples come from the Data-NM set. The dotted lines indicate 0 deg or deg/s. It is difficult to recognize distinctive patterns in these visualizations, but it appears as though there are more smooth, sinusoidal, shapes at the top (high contribution towards 'skilled').

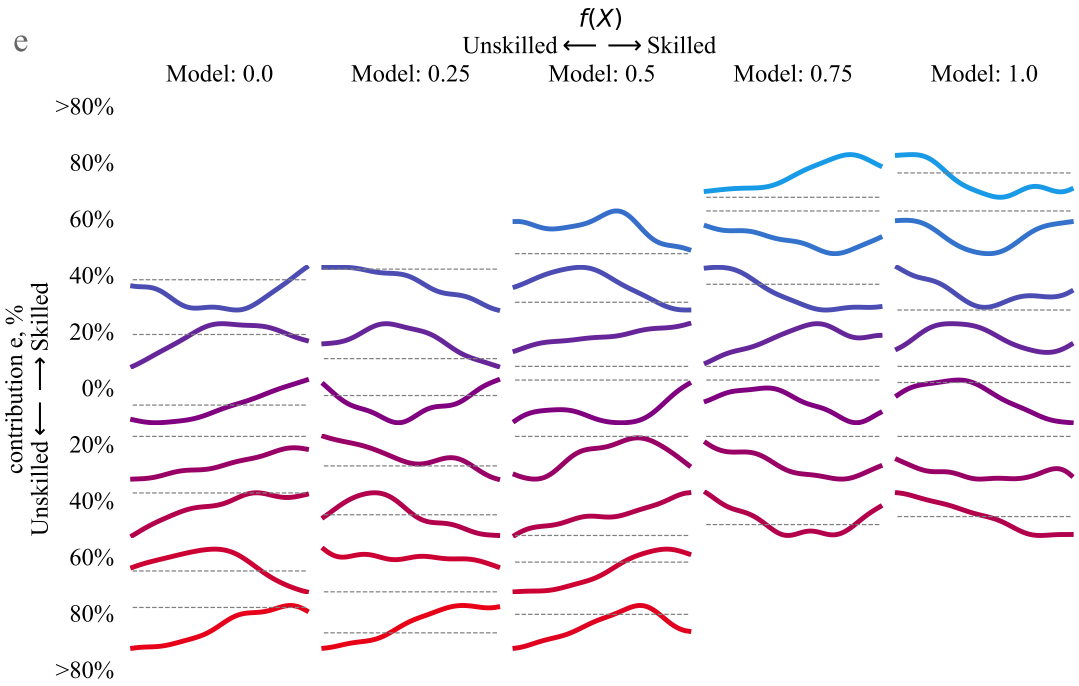


Figure D.3: Qualitative grid visualization of samples with different model output versus relative contribution of e.

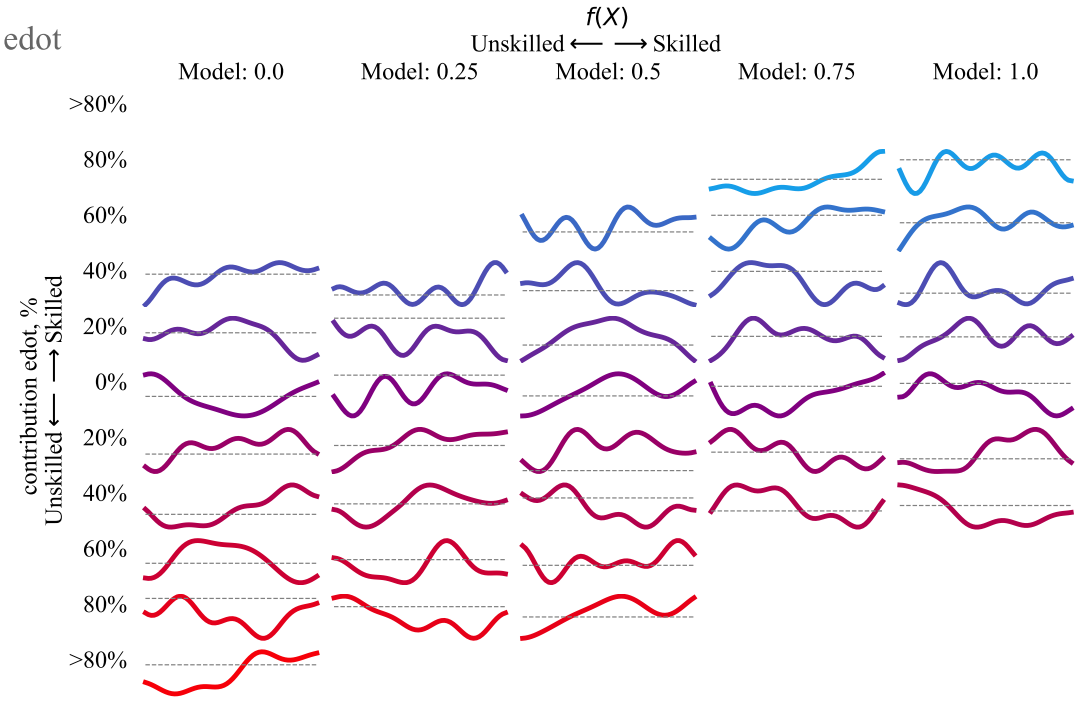


Figure D.4: Qualitative grid visualization of samples with different model output versus relative contribution of $\dot{e}$.

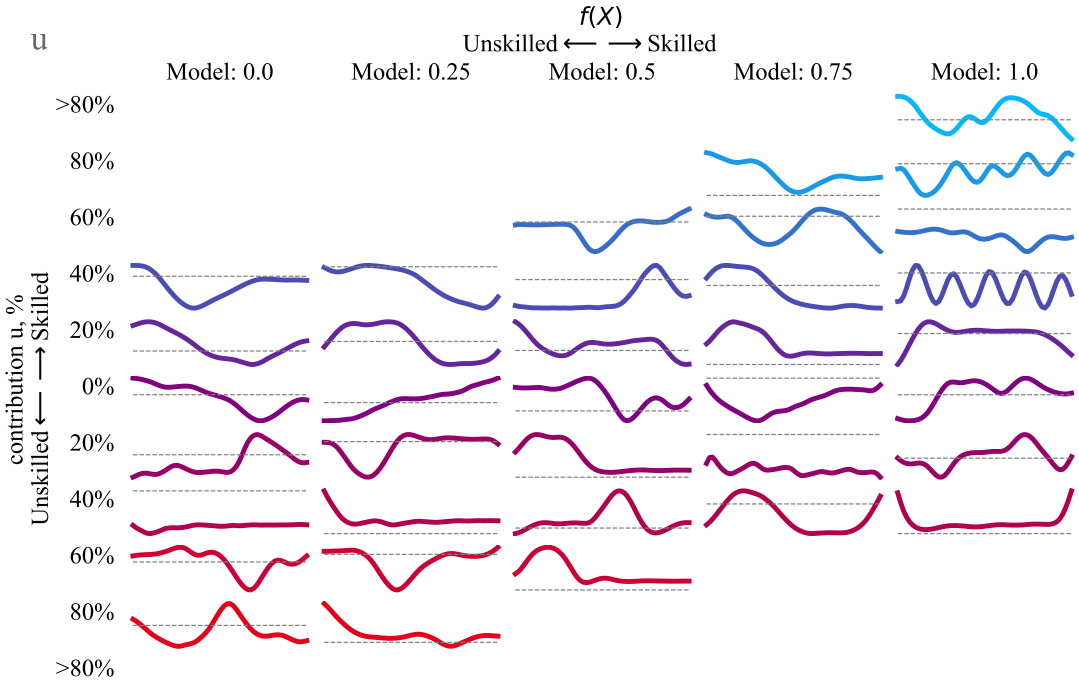


Figure D.5: Qualitative grid visualization of samples with different model output versus relative contribution of u.

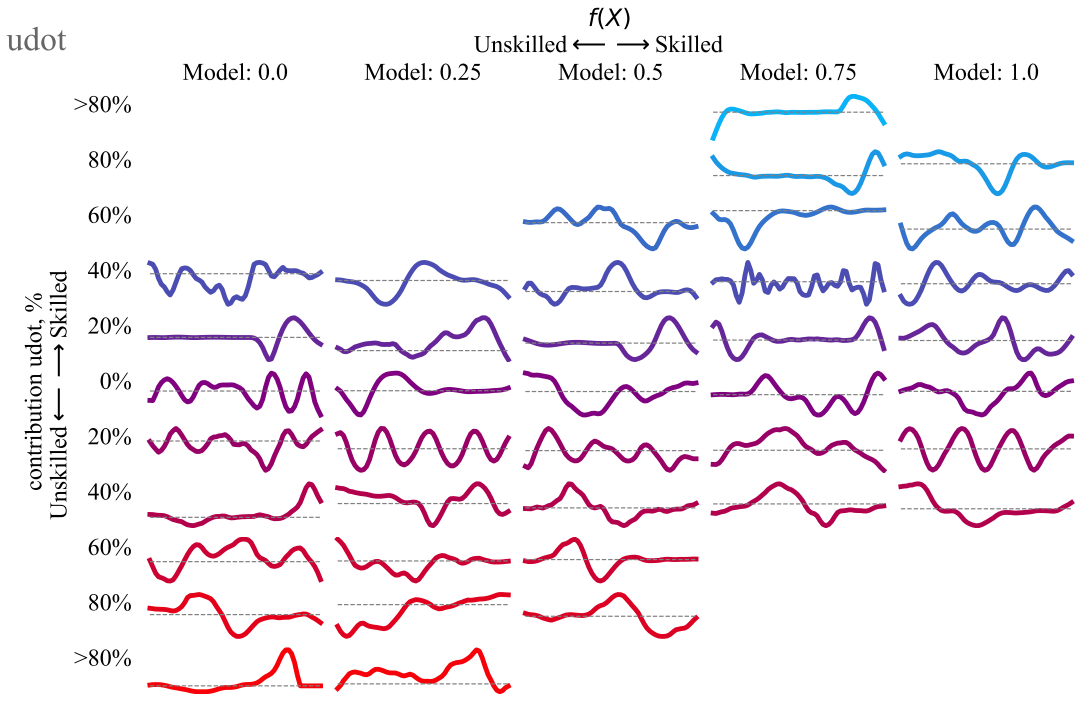


Figure D.6: Qualitative grid visualization of samples with different model output versus relative contribution of $\dot{u}$.

III

Preliminary Report (Already Graded)

1

Introduction

The idea of mathematically describing biological and social systems as circular feedback mechanisms has been around since the since late 40s, early 50s (Umpleby, 2005). After Wiener published his book in 1948 named *Cybernetics: or Control and Communication in the Animal and the Machine* a rise of cybernetics was seen in many academic fields. E.g. computer science, electrical engineering, artificial intelligence, robotics, family therapy, management, political science, sociology, biology, psychology, epistemology, music, etc (Umpleby, 2005).

Cybernetics as a tool to describe the characteristics of human *pilot* controlling a dynamic system have been developed since the late '50s (McRuer & Krendel, 1959). Since then these human control models have been extensively reviewed and enhanced to more complex and utilization-specific forms (Xu, Tan, Efremov, Sun, & Qu, 2017). This cybernetic approach of analyzing manual human control behavior has been successfully adopted to a broad variety of applications, such as human-in-the-loop control design of complex aircraft (Hess & Peng, 2018) or for the assessment of training of manual control skill (Pool & Zaal, 2016).

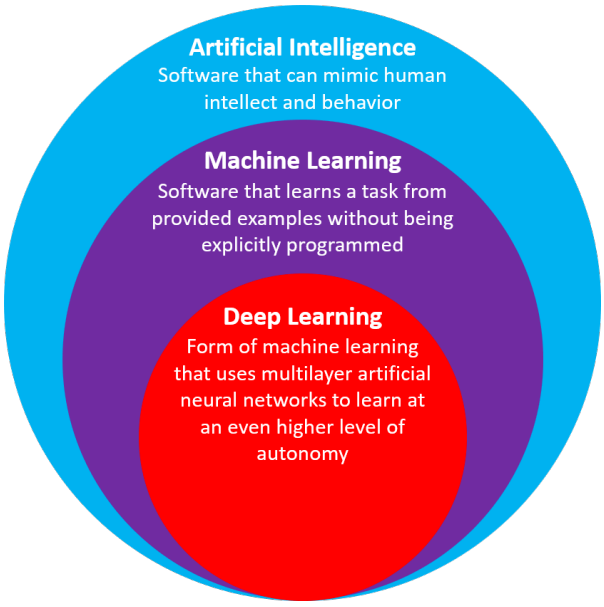


Figure 1.1: The relationship between artificial intelligence, machine learning, and deep learning. Figure based on NVIDIA blogpost:
https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Since the last decade there has been a huge growth in popularity of *artificial intelligence*. Just like cybernetics in the 50s, artificial intelligence applications span nearly every academic field now. Machine learning (a subset of artificial intelligence techniques, Figure 1.1) takes on many challenging tasks such as natural language processing (Wang & Gang, 2018), (medical) image recognition (Litjens et al., 2017), traffic prediction (Ramakrishnan & Soni, 2018), autonomous control (Al-Qizwini, Barjasteh, Al-Qassab, & Radha, 2017), and many more.

As artificial intelligence flourishes, the question arises if some of the tasks facilitated by cybernetics can also be performed by its self-learning counterpart.

This thesis will explore how machine learning can be used to evaluate pilot skill level. Although cybernetics already allows such evaluations, it often uses frequency domain analyses to predict human control behavior parameters (e.g. Pool and Zaal (2016)). This has the disadvantage of producing linear time invariant results that require lengthy time recordings to be accurate. Using artificial intelligence may allow faster recognition of patterns in the time domain, enabling time varying pilot evaluation feedback within seconds.

1.1. Introduction to Time Series Classification

Time series data are present in nearly every domain. Human activities, financial information, health recordings, weather readings, these all produce time series. The ability to recognize chunks of these time recordings as a certain class can allow us to monitor someone's health (Kampouraki, Manis, & Nikou, 2008), make financial market predictions (Fischer & Krauss, 2018), or recognize what activity a human is undertaking (Ordóñez & Roggen, 2016). The applications are endless. In this specific research report an effort will be made to apply Time Series Classification (TSC) to recognize human pilot skill level.

Before explaining what methods can be applied to perform TSC, first some general definitions will be introduced. The first three definitions are taken from Fawaz, Forestier, Weber, Idoumghar, and Muller (2019). To visualize these definitions, an example of a data set D containing two univariate time series X_i with their respective class vector Y_i is shown in Figure 1.2.

Definition 1: A univariate time series UTS, $X = [x_1, x_2, ..., x_T]$ is an ordered set of real values. The length of X is equal to the number of real values T.
Definition 2: A multivariate time series MTS, $X = [X^1, X^2, ..., X^M]$ consists of M different univariate time series with $X^i \in \mathbb{R}^T$.
Definition 3: A data set $D = \{(X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)\}$ is a collection of pairs (X_i, Y_i) where X_i could either be a univariate or multivariate time series with Y_i as its corresponding one-hot label vector.
One-hot label vector: For a data set containing K classes, the one-hot label vector Y_i is a vector of length K where each element $j \in [1, K]$ is equal to 1 if the class of X_i is j and 0 otherwise.
Definition 4: Time Series Classification deals with creating a mapping between time series X_i and label vector Y_i by learning from labeled training data from D.

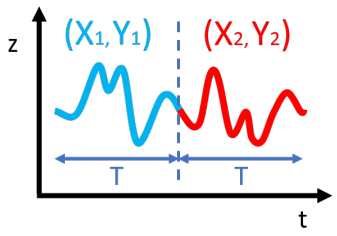


Figure 1.2: Example of an UTS data set D containing two pairs (X_i, Y_i)

There are many methods available to perform TSC. Susto, Cenedese, and Terzi (2018) describe two main branches of time series classification techniques: *feature-based* methods and *distance-based* methods.

Feature-based methods are based on the extraction of certain features from the time signal. A moving window extracts time series data from a signal and calculates representative features of this time series set. Commonly chosen features are for example: the mean, variance, minimum, maximum, or entropy (Susto et al., 2018). The idea is that these features characterize time series behavior that belongs to a specific class. By feeding these extracted features (or 'time series representations') and their respective label a classifier model can be taught to recognize classes based on their features. An example of such a classifier is a Support Vector Machine (SVM) (Evgeniou & Pontil, 2001). Other examples of time series representations are Singular Value Decomposition (SVD) (Li, Khan, & Prabhakaran, 2006) or symbolic representation (Lin, Keogh, Wei, & Lonardi, 2007) also refered to as Bag-of-Words (BoW).

Distance-based methods do not require feature extraction and can directly be applied to time series data. This approach is based around computing a measure of similarity between different time series. Typically this distance measure is dynamic time warping (DTW) (Keogh, 2002), after which the classification part can very effectively be done using the k-nearest-neighbours classifier (Cunningham & Delany, 2007).

A third method that could be distinguished are *ensemble-based* approaches. Here different classifiers are combined to achieve higher accuracy. A state-of-the-art example of such an ensemble classifier is the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) (Lines, Taylor, & Bagnall, 2018). This is an ensemble of 37 classifiers based on features extracted from both the time domain as well as the frequency domain. One disadvantage of HIVE-COTE is that it takes a lot of time to train, since it requires 37 classifiers to be trained to reach its high classification accuracy.

All of the 'traditional' machine learning TSC methods described above rely on substantial data preprocessing and feature engineering. More recently an effort has been made to design deep neural networks (a subset of machine learning techniques, Figure 1.1) for time series classification that have equal performance to state-of-the-art traditional TSC methods. A great example of this is given by Wang, Yan, and Oates (2017). In this paper three deep neural network architectures are compared against other state-of-the-art TSC approaches on 44 time series data sets. The three neural networks that are deployed are a classic feedforward network (also known as Multilayer Perceptrons (MLP)), a Fully Convolutional Network (FCN) (Long, Shelhamer, & Darrell, 2015), and a Residual Network (ResNet) (He, Zhang, Ren, & Sun, 2015). In the work by Wang et al. (2017) it is found that these latter two networks (FCN and ResNet) are actually able to outperform other state-of-the-art methods without requiring any data preprocessing or feature engineering. Moreover, these specific deep neural network architectures were designed for image recognition and yet still showed such promising results on a completely different (TSC) task. An explanation of how neural networks work will be provided in Chapter 3.

Automatic feature extraction is a great advantage that deep learning algorithms have over traditional machine learning algorithms. In the past, machine learning practitioners have spent months, years, or even decades manually constructing feature sets for classification of data (Patterson & Gibson, 2017). Artificial neural networks have the ability to automatically extract these features and learn what indicators to look for to accurately classify data themselves.

A major drawback of deep neural networks is that they are considered black box models. That is, although they can show exceptional performance in many tasks, it can be difficult to identify why they make certain decisions. However, the works of Wang et al. (2017) and Fawaz et al. (2019) show that specific types of neural networks can actually overcome this problem and an 'explainability' aspect can be added to the network's output. On top of that, Lundberg and Lee (2017) introduced a unified approach to crack open the black box of all sorts of deep neural network architectures.

Given some of the tremendous advances made in the field of deep learning (Esteva et al. (2017), Silver et al. (2017), Lim, Son, Kim, Nah, and Lee (2017), and more recently by Hannun et al. (2019), Kim, Zhou, Philion, Torralba, and Fidler (2020), Saito, Simon, Saragih, and Joo (2020)), their proven effectiveness in TSC (Wang et al. (2017), Fawaz et al. (2019), Wang, Chen, Hao, Peng, and Hu (2019), Acharya et al. (2017)), and their added benefit of automatic feature extraction, the decision was made to limit the scope of this thesis to deep learning TSC methods.

1.2. Introduction to Human Pilot Modeling

Before introducing basic human pilot modeling concepts, it will first be explained why pilot modeling is relevant for this TSC research. Plain and simply put: human pilot modeling will allow simulation of pilot control behavior to generate more training data for the machine learning algorithm. There are two reasons why generation of additional pilot data may be interesting. First, machine learning algorithms often benefit from having a larger set of training data available. Second, by making use of human pilot modeling to generate pilot data there will be more control over the training data that is supplied to the machine learning algorithm.

Generating more training data based on available training data to improve the performance of classification systems is also referred to as *data augmentation*. This technique is very commonly applied to computer vision tasks, a survey by Shorten and Khoshgoftaar (2019) describes many image data augmentation techniques and their effectiveness. However, there have also been data augmentation methods developed specifically for time series classification. In a survey by Wen et al. (2021) some of these methods are explained and reviewed. There are practically two ways to perform data augmentation: new data can be generated by slightly altering the existing data, or new data can be completely synthetically generated (making sure that it resembles the original data). This latter approach is where human pilot modeling comes into play.

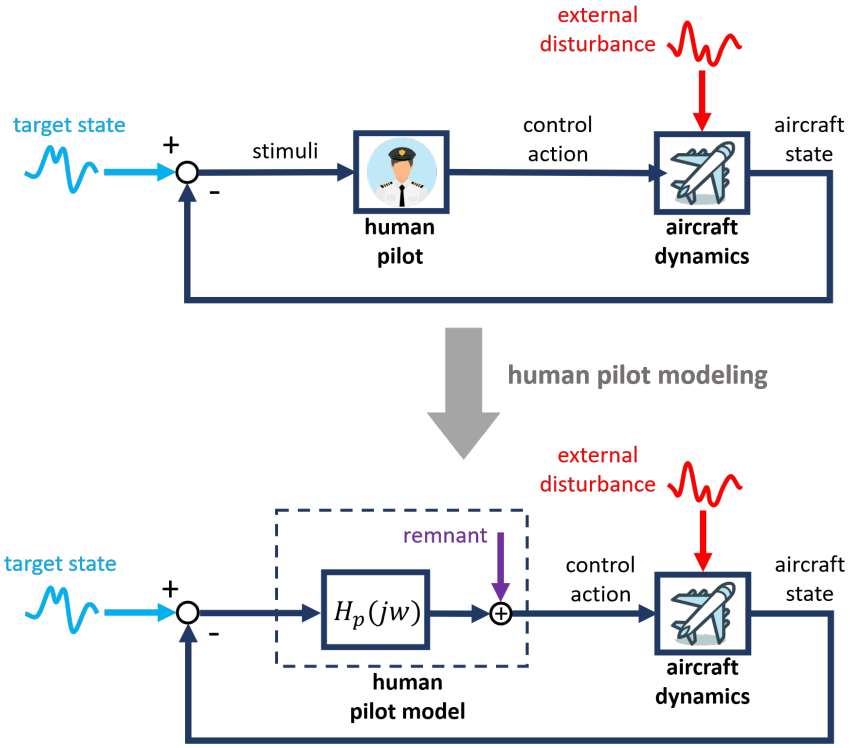


Figure 1.3: Schematic illustration of human pilot modeling

Human pilot models describe human manual control behavior response to a specific task. By identifying pilot control dynamics a mathematical model can be composed that describes their control response to certain stimuli. These models can help to understand human control behavior, but can also quantify human behavior through system- evaluation and simulation.

Modeling human pilot control behavior has proven to be most successful for continuous and stationary control tasks (tracking) (Pool, Pais, Vroome, van Paassen, & Mulder, 2012). For such tasks human control behavior can accurately be described by quasi-linear pilot models as proposed by McRuer et al. (1965). These quasi-linear models consist of linear transfer functions that describe the human response to perceived inputs, and a *remnant* that accounts for non-linear human control behavior. By letting human pilots perform a control task in a controlled environment, their control actions can be

analysed to compute the pilot model parameters that make up the linear transfer functions.

A non-technical schematic illustration of the pilot modeling process is shown in Figure 1.3. In the top half of this figure it can be seen how a human pilot outputs control actions based on stimuli they receive (in this case from a display). The control action then causes a new aircraft state through the aircraft dynamics. This new state is not fully determined by the control action, but slightly unpredictable due to external disturbances (e.g. turbulence). The new aircraft state is fed back to the start of the loop where it is compared to the target state. The display will show the pilot the updated discrepancy between the desired state and actual state to which the pilot will respond, and so the loop continues. In the bottom half of the figure it can be seen how the human pilot is replaced by a quasi-linear model consisting of a linear transfer function $H_p(j\omega)$ and a remnant. The goal is to estimate the model parameters such that the model behavior replicates the human behavior. A more detailed and technical explanation of human pilot modeling will be provided in Chapter 2.

1.3. Problem Statement

This section will first provide a motivation for the research that is conducted throughout this report. This motivation is based on current developments of state-of-the-art methods and the academic knowledge gap to the research topic. Based on the motivation a research question (and sub-questions) will be formulated. These questions will be posed such that their answers will bridge the academic knowledge gap identified in the motivation. Lastly, an explicit research objective will be stated.

Motivation In recent years there have been many researches attempting to classify human control behavior based on measurements of both task specific performance parameters as well as physiological signals. The control task that appears to be researched most intensively is that of a human controlling a car. A survey by Marina Martinez, Heucke, Wang, Gao, and Cao (2018) elaborates on the applicability of *driving style recognition* for energy management, driving safety, and driving assistance. This survey also puts a particular emphasis on machine learning as a tool for the task. Similarly, Jain, Singh, Koppula, Soh, and Saxena (2016) investigates the use of machine learning to anticipate *driver activity*, Saleh, Hossny, and Nahavandi (2017) uses machine learning to classify *driving behavior*, and Tango and Botta (2013) applies machine learning to detect *driver distraction*. Although these researches are predominantly about humans operating cars, there are also examples of machine learning algorithms classifying control behavior of human pilots in aircraft. Both Xi, Law, Goubran, and Shu (2019) and Nittala et al. (2018) propose machine learning methods to predict pilot workload, the latter also incorporated a skill level classification algorithm. Specifically this skill level classification using machine learning is a research topic that remains relatively untouched.

Although there has been a substantial amount of research done regarding human control behavior classification using machine learning, most of these methods require an extensive set of input parameters (collected by a large set of sensors). For example the mentioned papers concerning humans driving cars use sensor such as: (infrared) face cameras, eye tracking devices, outside cameras, GPS, and numerous inertial measurement sensors. The aircraft pilot examples both use electrocardiograms (ECG) to measure the pilot's heart rate. Evidently, applying these machine learning algorithms in a real life application would require a lot of additional sensors to be added to the operated vehicle.

Therefore, this research will attempt to introduce a novel *pilot skill level classification* method using deep learning with only readily available task related time signals (e.g. pilot control output, tracking command, and tracking error). Such a method could be used in pilot training environments to quantify a pilot's skill level, removing the need for subjective qualitative assessment. It could also be used in an online setting to enable a scalable level of autonomy; the autopilot only 'helps' as much as it deems necessary. Or, if there is no autopilot, a simple warning signal could be given if a deteriorating skill level is measured. Due to the few (and rudimentary) input signals required, this method would not need any additional sensors to be added to the cockpit of most modern aircraft. Another advantage of this choice of input signals is that the method should be easily transferable between different vehicles.

Research question Thus far some fundamental information has been shared to illustrate the goal of this thesis. Moreover, a motivation has been provided to clarify how the findings of this research could be useful in real life applications. To further constrain exactly what will be investigated in this research (and what will be left out of scope), a set of research questions has been composed.

The main research question that this thesis will ultimately aim to answer is the following:

Main research question

How can deep learning be used to identify pilot skill level?

As has already been discussed in the introduction there are more dimensions to this question than immediately apparent. The foremost and most obvious portion of this question that has to be answered is *how can deep learning be used to classify time series data?*. However, this research will also attempt to overcome some of the drawbacks that come with artificial neural networks. Namely, the following two disadvantages will be tackled: *too little data availability deteriorates training performance* and *the black box nature of deep learning gives little insight into the algorithm's process*.

The above mentioned dimensionalities of the main research question reap the following sub-questions:

Research sub-questions

1. **How can artificial intelligence be used to classify time series data?**
   (a) What algorithms can be used to perform this task?
   (b) How must the training data be structured?
   (c) What influence do hyperparameters have on the classification performance?
   (d) What is the optimal performance achieved with the proposed method?
2. **How can pilot modeling be used to generate additional training data?**
   (a) What type of model can be used to simulate pilot behavior?
   (b) What influence do pilot model parameters have on classification performance?
   (c) What is the added benefit of the proposed data generation method?
3. **How can an explainability component be added to the classification model?**
   (a) What explainability methods are available?
   (b) What input parameters influence the classification model's output and how?
   (c) How can the proposed explainability method be used to interpret the classification model's decision making?

Research objective Now that the research questions have been defined, the objective of this research can be formulated. The research objective will summarize the research questions into actions that will compose the goal of this research project.

Research objective

To effectively classify the skill level of pilot control behavior, using state-of-the-art deep learning time series classification algorithms in combination with a cybernetic data augmentation method, whilst providing transparency to the results by making the trained algorithm interpretable.

1.4. Report Overview

This report effectively consists of two parts: a literature study and a preliminary experiment. The first part, the purely literature based portion, of this report is found in Chapter 2 and Chapter 3. These chapters introduce the three pillars, reflected in the research questions, on which this research thesis is based. They are structured as follows.

First, in Chapter 2 human pilot behavior data and cybernetic pilot modeling as a means for data

augmentation are explored. This chapter will introduce some basic human behavior principles and will explore how pilot modeling techniques may be used for data augmentation to achieve increase machine learning performance. The intent of this chapter is to answer sub-question 2 (a) and the non-empirical part of sub-question 1 (b).

Second, in Chapter 3 the fundamentals of artificial neural networks are explained, after which a more detailed description will be given of state-of-the-art networks that are specifically designed for time series classification. Additionally, Section 3.4 will elaborate on explainable artificial intelligence. Its importance is emphasized and several implementation techniques are described. This chapter aims to answer sub-question 1 (a) and sub-question 3 (a).

After the literature study is conveyed and an overview of available methods is collected, the second part of this report will test the gathered information from literature in a preliminary experiment. The results of this preliminary experiment are provided in Chapter 4. The goal of this preliminary experiment is to get an understanding of what the answers to the remaining research questions may be, this will help to compose a detailed research plan for the main phase of this thesis.

2 Human Pilot Data

The fundamental working principle of machine learning is to make an algorithm learn from examples. This is based on the premise that there are examples (training data) available for the machine learning algorithm to learn from. The content of these data is very important for the performance of the algorithm; a perfect machine learning model that is provided poor training data will also produce poor results. Concurrently, there is also the problem that sometimes there is an inefficient amount of representative training examples available.

The objective of this chapter is twofold: in Section 2.1 a foundation will be laid for what type of training data is required for this research and in Section 2.2 a method to overcome shortage of representative training data will be proposed.

2.1. Data Specifications

As has been stated in the introduction, the goal of this research is to identify pilot *skill level* using machine learning. Before describing how this identification can be performed, it must first be explained what is meant by *skill level*. Once this definition has been clarified, an explanation must be given of what this means for the requirements of the data set that the machine learning algorithm will be trained with.

2.1.1. Different Types of Control Behavior

When a human carries out a control task, they execute control performance that fits in a pattern of human control behavior. In a paper by Rasmussen (1983) three typical levels of performance are distinguished: skill-based behavior, rule-based behavior, and knowledge-based behavior. Figure 2.1 shows a depiction of these three behavior levels and how they interact.

Skill-based behavior represents human control behavior that takes place without conscious control. The sensory information that is perceived during skill-based behavior are continuous signals, e.g. visually perceived information from a display, or motion feedback from the vestibular system. Following a certain intention the human will automatically perform continuous control actions as a response to these signals.

Rule-based behavior relates to task execution that is controlled by stored rules or procedures. These rules or procedures may have been derived from past experience or from someone else's expertise as instructions. Actions taken from rule-based behavior follow from signs that are recognized by the human, e.g. knowing to step on the brakes of a car when a traffic light jumps to red. The border between rule-based behavior and skill-based behavior is not always definite, but depends on the level of training of the human in question. However, a big difference between the two is that in rule-based behavior the person can understand and explain why they took a specific action. Contrarily, in skill-based behavior actions are taken subconsciously and cannot be pinpointed to a specific source of information.

Knowledge-based behavior is triggered during unfamiliar situations in which no know-how or rules for control are available. In such situations the performance is moved to the highest conceptual level

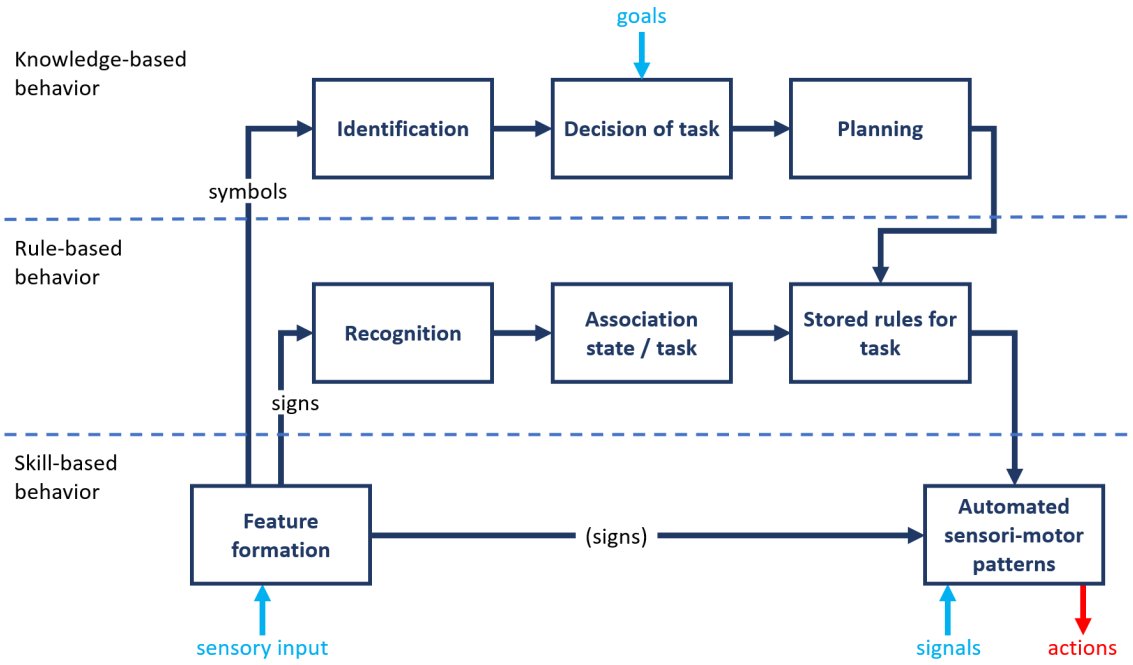


Figure 2.1: The three levels of human control behavior, as defined by Rasmussen (1983)

where behavior is goal-controlled. During knowledge-based behavior information is perceived as symbols which are used to develop a plan to reach the specified goal. In the development of the plan multiple options are considered and tested against the goal, physically by trial and error, or conceptually by understanding the functional properties of the controlled system.

Given the descriptions of the three different performance levels of human control behavior, it can now be explained what is meant by pilot skill level. Namely, *pilot skill level is a qualitative indication of the proficiency of a pilot's skill-based behavior*. A high skill level means that a pilot has a lot of experience in executing a certain task and has therefore trained their automated responses to accurately react to the task related continuous signals. A low skill level means that a pilot has little or no experience in executing a task and therefore executes poor skill-based behavior.

2.1.2. Pilot Skill Level Data

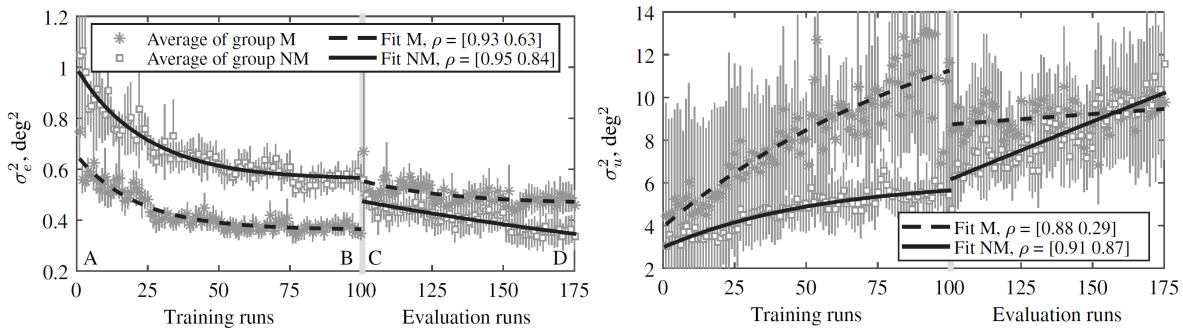
To be able to effectively deploy machine *learning* algorithms, there must be a source of data to *learn* from. Or, following the third definition given in Section 1.1, there must be a (sufficiently large) data set D containing pairs (X_i, Y_i). For this research specifically that means that there must be time series recordings of human pilot behavior X_i and their respective categorical skill level label Y_i. If there are two levels of skill that the classifier must be able to distinguish (e.g. 'novice' and 'expert'), then there must also be examples of both these skill levels provided to the machine learning algorithm. As explained in the previous section, a pilot's skill level for a task is determined by the amount of experience said pilot has in that (skill based) task. Therefore, a data set D is required that contains time series (and the associated label) of both task-naive pilots, as well as trained pilots.

This data set could either by collected by conducting an experiment, or could alternatively be taken from an existing experiment that has already taken place. The most obvious type of experiment to find such a data set, that adheres to the specifications described in the previous paragraph, would be an experiment that researched training of skill-based control behavior. Luckily, the section Control and Simulation of Aerospace Engineering at TU Delft is precisely the place to look for such experiments.

In a research paper called 'Effects of Simulator Motion Feedback on Training of Skill-Based Control Behavior' Pool et al. (2016) conducted an experiment in SIMONA Research Simulator at Delft University of Technology to quantify the effects of simulator motion feedback on the training of skill-based human

operator control behavior. In this experiment a group of 24 fully task-naive participants were trained in a compensatory pitch attitude tracking task. Here 'compensatory' means that the participants could only read the *error* between the desired pitch angle and the actual pitch angle on a display; why this is important will be explained in Section 2.2.2. Divided over two groups, one group *with* motion feedback ("moving-base") and one group *without* motion feedback ("fixed-base") the participants performed 100 training runs of the pitch tracking task. After the 100 training runs the two groups switched motion feedback setting and performed another 75 evaluation runs. This change of motion feedback setting was done to investigate to what extend trained skill can be transferred to a different setting.

To visualize the learning curve of these participants, consider the graphs shown in Figure 2.2 (Pool et al., 2016). In Figure 2.2a the tracking error variance σ_e^2 per tracking run is shown, and in Figure 2.2b the pilot control output variance σ_u^2. The grey asterisk and square markers indicate the average variance for the group *with* motion feedback (M) and the group *without* motion feedback (NM), respectively. The gray bars indicate the 95% confidence intervals of the mean data. Lastly, the learning curve is quantified by fitting an exponential learning curve model (Eq. (C.6)). The learning curve models are indicated by the black (dashed/solid) lines. Pearson's correlation coefficient are given in the legend for both the training and evaluation phase as $\rho = [\rho_{training}\ \rho_{evaluation}]$, as an indication of the quality of the fitted learning curve. From these graphs it immediately becomes apparent that, as the subjects (pilots) gain more experience with the tracking task, the tracking error variance decreases and the control output variance increases. These changes in control behavior indeed indicate that the pilot skill level increases with practice.



(a) Tracking error variance per experiment run

(b) Pilot control output variance per experiment run

Figure 2.2: Average tracking error variance and pilot control output variance with fitted learning curves. Image taken from (Pool, Harder, & van Paassen, 2016)

The initial 100 training runs from (Pool et al., 2016) are exactly the type of data set that was described at the start of this section as a requirement for this TSC research. Namely, it contains human control behavior recordings of both fully task-naive participants (at the start of their 100 runs), as well as experienced participants (at the end of their 100 runs). The only thing that still had to be added to these data before it could be fed to a machine learning algorithm are the labels Y_i of each recording X_i. Different methods of labeling the data and their pros and cons are discussed in Section 4.2.3.

2.2. Data Augmentation

In the introduction of this report it was already suggested that modeling human manual control behavior may be useful to generate additional training data for the machine learning algorithm. Generating additional training data to improve machine learning performance is also known as *data augmentation*. Although data augmentation is primarily used in computer vision tasks, it also has an proven effectiveness in time series classification (e.g., Wen et al. (2021) shows an improvement of almost 2% increased accuracy on an otherwise exhausted training set). This section will further elaborate on data augmentation, why it is useful, and how it may be achieved through human pilot modeling.

As will be discussed more thoroughly in Chapter 4, deep learning algorithms often cycle through the training data multiple times while learning. Every cycle through the complete set of training data is called an *epoch*. By having multiple epochs the training data is more efficiently used and may produce better accuracy of the trained algorithm. However, if the same data is presented too many times a phe-

nomenon called *overfitting* may occur. Overfitting means that the trained algorithm becomes extremely good at recognizing the training data, but can no longer accurately classify data that are outside of its training set (whilst, of course, accurately classifying unseen data is exactly the goal). There are many ways to reduce overfitting, some of these will be discussed in Chapter 3.

One way of overcoming overfitting is increasing the amount of labeled training data. This way less repetitions (epochs) of the same learning material are required to train the deep learning network. However, it is often the case that there is not any more training data available. In this situation *data augmentation* can be used to artificially inflate the training data set size by either *data warping* or *oversampling* (Shorten & Khoshgoftaar, 2019).

Here data warping means to transform existing data such that their label is preserved. In computer vision tasks this can be achieved by augmentations such as geometric transformations, color changes, or random distortions (some image data warping examples are shown in Figure 2.3). Similarly, in TSC tasks data warping is accomplished by augmentations such as scaling, cropping, rotating, time warping, or jittering can be applied (examples of this are shown in Figure 2.4). A lot of these data warping operations are more often used on image data than on time data. This is because for images a visual comparison can confirm that the augmentation (e.g. rotation) did not alter the image's class, whereas for time series data this is not the case (Fawaz, Forestier, Weber, Idoumghar, & Muller, 2018). However, cropping (or 'window slicing') is an example of data warping augmentation that can very effectively be used in TSC (this will be applied, and discussed, in Chapter 4). Consecutively sliced windows can also be overlapped to to even further increase the amount of labeled training samples.

Oversampling means to generate new synthetic instances of training data that adhere to a certain label. This can be done by mixing existing data, performing feature space augmentations, or by using generative adversarial networks (GANs) (Shorten & Khoshgoftaar, 2019). Examples of works researching time series data modeling with GANs are Esteban, Hyland, and Rätsch (2017) and Yoon, Jarrett, and van der Schaar (2019). This thesis will research the effectiveness of *oversampling* data augmentation by means of cybernetic pilot modeling to simulate additional time series data of human control behavior.

2.2.1. Pilot Modeling Techniques

Being able to model pilot control behavior allows the analysis of closed loop pilot-vehicle systems. Such integrated closed loop system with human control behavior have many applications in the field of aerospace engineering. Pilot-vehicle systems can for example be utilized to assess aircraft handling qualities (e.g. Hess (1995)), or to design flight instruments (e.g. McRuer, Weir, and Klein (1971)), or to evaluate the fidelity of flight simulators (e.g. Steurs, Mulder, and van Paassen (2004)). These are just some of the many examples of applications that drive the need for development of accurate pilot behavior models.

The aim of this research is not to explore novel methods for state-of-the-art pilot simulations, but rather to employ an existing proven pilot model and test its effectiveness in a new application: data augmentation. Therefore, it is worth mentioning that this section will not dive into great detail about all different types of pilot modeling techniques. Such a review of available methods of control model for human pilot behavior are provided by Lone and Cooke (2014) and Xu et al. (2017).

As mentioned in Section 2.1.2 the data for this preliminary research thesis will be taken from the experiment by Pool et al. (2016). In this experiment a multimodal quasi-linear human operator model was used to model participants' control dynamics. For every run of each participant a set of model parameters is composed that best describe their control behavior. These model parameters are then used as a measure to identify training of skill. The same compensatory tracking task used in this experiment had already been successfully used to identify multimodel human control behavior in earlier experiments (e.g. Zaal, Pool, de Bruin, Mulder, and van Paassen (2009), Pool, Zaal, van Paassen, and Mulder (2010), and Nieuwenhuizen, Mulder, van Paassen, and Bülthoff (2013)).

Precisely because (multimodal) quasi-linear pilot models are excellent at simulating human control behavior during a compensatory tracking task, and because the research by Pool et al. (2016) has found fitting model parameters for each of its participants' tracking runs, the decision was made to use a quasi-linear pilot model as a tool for data augmentation.

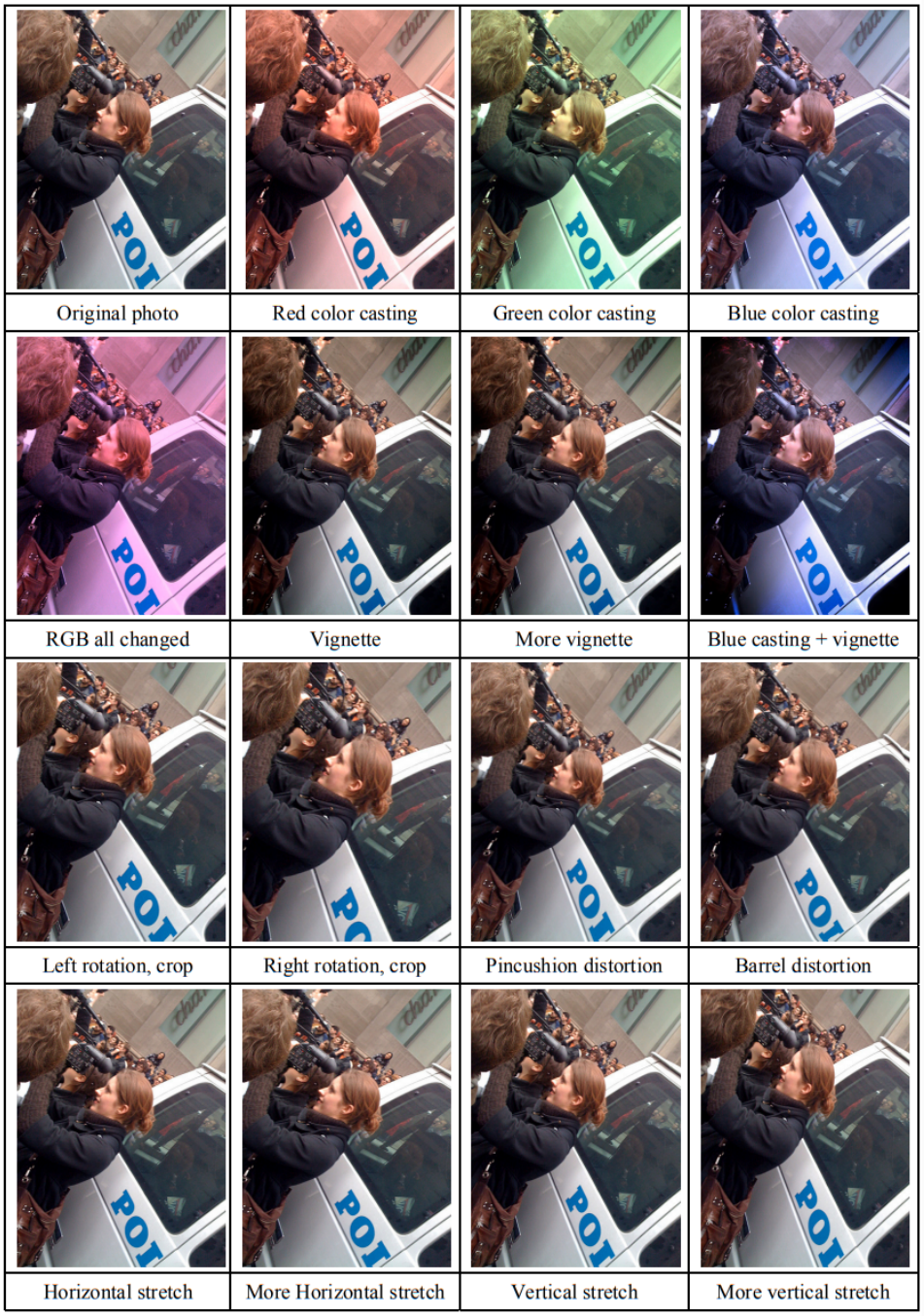


Figure 2.3: Examples of data augmentation techniques for computer vision. Image taken Wu, Yan, Shan, Dang, and Sun (2015)

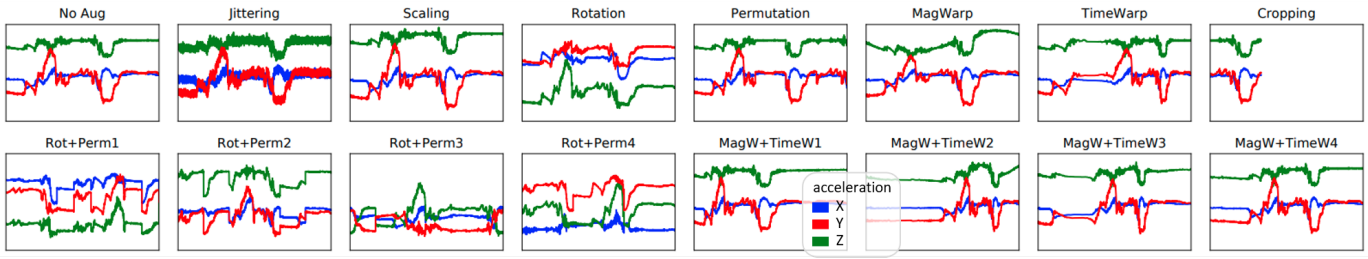


Figure 2.4: Examples of data augmentation techniques for time series. The blue, red, green represent X,Y,Z signals from an accelerometer, respectively. Image taken Um et al. (2017)

2.2.2. Quasi-Linear Pilot Model

In the experiment by Pool et al. (2016) the participants were given a compensatory pitch tracking task as shown in Figure 2.6. The controlled element dynamics H_{θ,δ_e} in this experiment were a reduced-order linearized model of elevator-to-pitch dynamics of a Cessna Citation I. This model was taken from Zaal et al. (2009) and is given by Eq. (2.1). The stick inputs given by the participant were scaled with a stick gain K_s such that 1 deg of stick deflection gave an elevator deflection δ_e of -0.2865 deg.

$$H_{\theta,\delta_e}(s) = 10.62\frac{s+0.99}{s\,(s^2+2.58s+7.61)} \tag{2.1}$$

In a compensatory pitch tracking task the participant must continuously attempt to minimize a pitch tracking error e which can be read from a visual display like the one shown in Figure 2.5. The desired pitch angle θ that the pilot must follow is given by the *target* forcing function signal f_t. Meanwhile, an additional *disturbance* forcing function signal f_d is inserted before the controlled dynamics. By only displaying e and having an unpredictable (quasi-random) f_t (and f_d) the pilot is forced to adopt to a purely compensatory control strategy (McRuer & Jex, 1967). Precisely this behavior can successfully be modeled by the quasi-linear model introduced by McRuer et al. (1965).

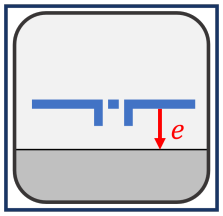


Figure 2.5: A compensatory display of a pitch tracking task, as was provided to the participants

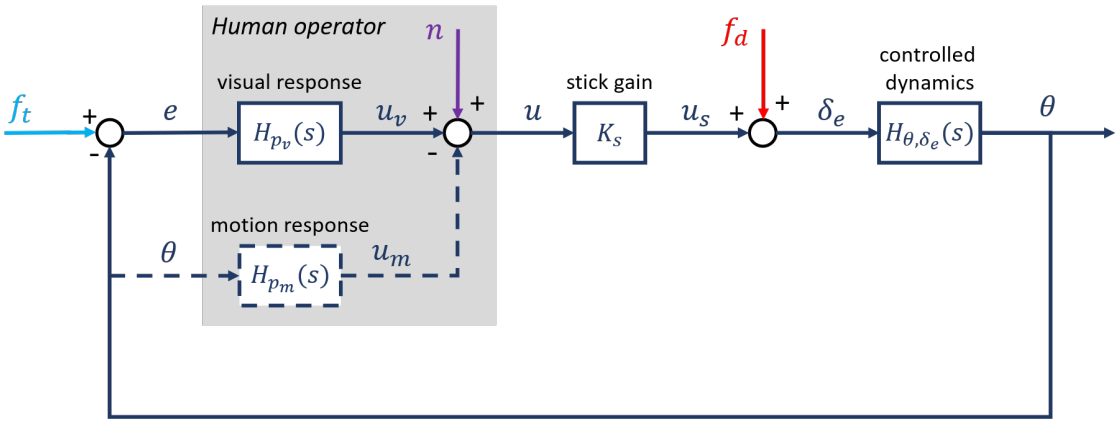


Figure 2.6: Schematic representation of the compensatory pitch attitude tracking task

The block marked as '*Human operator*' in Figure 2.6 depicts the multimodal quasi-linear pilot model used in the experiment. This model consists of parallel visual and vestibular motion channels and a remnant (n). The motion channel was only incorporated if motion feedback was provided to the participant. The pilots' responses to the visual tracking error e and the physical pitch angle θ are calculated by the response functions $H_{p_v}(s)$ and $H_{p_m}(s)$, respectively. Since the task includes both target-following (f_t) and disturbance-rejection (f_d) the separate contributions of the visual response and motion response could be investigated (Stapleford, Peters, & Alex, 1969).

The **visual response** of the human operator model $H_{p_v}(s)$ was given by Eq. (2.2). This equation consists of a lead^2-lag equalization characteristic, a human operator response delay, and a model for neuromuscular dynamics $H_{nm}(s)$ given by Eq. (2.3). Because human operators generate lag at frequencies below the short-period mode's natural frequency of Eq. (2.1), but exert lead equalization at higher frequencies, a quadratic lead term had to be added to Eq. (2.2) to model such equalization dynamics (Pool et al., 2011).

In total the visual response model (Eq. (2.2)) contains six model parameters: the visual gain K_v, the visual lead time constant T_{lead}, the visual lag time constant T_{lag}, the visual time delay τ_v, and the neuromuscular dynamics modeled as a second-order mass-spring-damper (McRuer et al., 1965) with neuromuscular frequency ω_{nm} and neuromuscular damping ratio ζ_{nm}.

$$H_{p_v}(s) = \underbrace{K_v \frac{(T_{lead}s+1)^2}{T_{lag}s+1}}_{\substack{\text{pilot}\\ \text{equalization}}} \underbrace{e^{-s\tau_v}}_{\substack{\text{response}\\ \text{delay}}} \underbrace{H_{nm}(s)}_{\substack{\text{actuation}\\ \text{dynamics}}} \tag{2.2}$$

$$H_{nm}(s) = \frac{\omega_{nm}^2}{s^2 + 2\zeta_{nm}\omega_{nm}s + \omega_{nm}^2} \tag{2.3}$$

The **motion response** of the human operator model $H_{p_m}(s)$ was modeled as pilot's response to his vestibular motion sensitivity as proposed by van der Vaart (1992) in his *multi channel model* and Hosman and Stassen (1999) in their *descriptive model*. The pilot motion response is then given by Eq. (2.4). Here the dynamics of the semicircular canals (SCC), the vestibular sensors that allow humans to sense angular acceleration, are modeled as $H_{scc}(s)$ shown in Eq. (2.5).

Two additional model parameters are introduced by the motion response model (Eq. (2.4)): the motion gain K_m and the motion time delay τ_m. The SCC model (Eq. (2.5)) is assumed to be fixed and equal for every participant, here $T_{ssc_1} = 0.11$ s, $T_{ssc_2} = 5.9$ s, and $T_{ssc_3} = 0.005$ s, these values are found using experimental measurements of human motion perception thresholds (Hosman & van der Vaart, 1978).

$$H_{p_m}(s) = s^2 H_{scc}(s)\, K_m e^{-s\tau_m} H_{nm}(s) \tag{2.4}$$

$$H_{scc}(s) = K_{scc}\frac{1 + T_{scc_1}s}{(1 + T_{scc_2}s)(1 + T_{scc_3}s)} \approx K_{scc}\frac{1 + T_{scc_1}s}{1 + T_{scc_2}s} \tag{2.5}$$

The **remnant** of the human operator model n represents the cumulative sum of all nonlinear contributions to the human manual control behavior (Pool et al., 2012), i.e. the portion of control behavior that is not captured by the linear transfer functions $H_{p_v}(s)$ and $H_{p_m}(s)$. In the paper by McRuer and Jex (1967) it is said that, in ascending order of importance, the sources of remnant are due to the following: 1) *pure noise injection* by the human pilot, 2) *nonlinear operations* such as indifference thresholds, control output and rate saturation, 3) *nonsteady pilot behavior* i.e. the parameters of the quasi-linear pilot model can only be defined meaningfully as averages over certain time lengths, but in reality pilot behavior has significant time variation during tracking.

The experiment by Pool et al. (2016) did not need a remnant model since the quasi-linear pilot model was used to *evaluate* the individual tracking runs of participants (by fitting the human operator model parameters to the data from the individual tracking runs), rather than used to *simulate* the participants' pilot behavior. Therefore an implementation of remnant modeling was taken from a different research by van der El et al. (2019). In this research the remnant is modeled as filtered white noise (i.e. colored noise).

As proposed by Levison et al. (1969) the first-order low-pass filter shown in Eq. (2.6) (with gain K and lag time-constant T_l) is used to capture the equivalent remnant spectrum injected at the error. A relationship between the power-spectral density function of the remnant injected at the error $S_{nn_e}(j\omega)$ and the power-spectral density of a white noise source $S_{ww}(j\omega)$ is defined as shown in Eq. (2.7). In a compensatory control task the power-spectral density function of the remnant injected at the error $S_{nn_e}(j\omega)$ can be approximated at the target frequencies ω_t and at the disturbance frequencies ω_d using Eq. (2.8) (derivation provided by van der El et al. (2019)). Here S_{uu_n} is the pilot control output power-spectral density due to remnant n, S_{uu_d} due to disturbance f_d, and S_{uu_t} due to target f_t. $S_{f_df_d}$ and $S_{f_tf_t}$ are the power-spectral density functions of the disturbance and target signal, respectively.

Now the filter gain K and lag time-constant T_l can be determined by using a least squares cost function to fit the estimated remnant Eq. (2.8) to the remnant model Eq. (2.7). As a last step the remnant can now be included in the pilot simulation by injecting filtered white noise (filter from Eq. (2.6) with found K and T_l) at the error.

$$H_n(j\omega) = K\frac{1}{1+T_l j\omega} \tag{2.6}$$

$$S_{nn_e}(j\omega) = |H_n(j\omega)|^2 S_{ww}(j\omega) \tag{2.7}$$

$$\begin{aligned} S_{nn_e}(j\omega_d) &= \frac{S_{uu_n}(j\omega_d)}{S_{uu_d}(j\omega_d)} S_{f_d f_d}(j\omega_d) \\ S_{nn_e}(j\omega_t) &= \frac{S_{uu_n}(j\omega_t)}{S_{uu_t}(j\omega_t)} S_{f_t f_t}(j\omega_t) \end{aligned} \tag{2.8}$$

The implementation, accompanied with figures and quantitative results, of above explained human operator model will be provided in Section 4.3.

2.3. Chapter Takeaways

This chapter explained two things: 1) what data will be used for this machine learning thesis and 2) how a pilot model may be used to generate additional data.

The selected data that will be used to train the machine learning algorithm are data from an experiment by Pool et al. (2016). In this experiment, a group of fully task-naive participants performed 100 training runs in a compensatory tracking task. This data set is ideal for this thesis, because it contains pilot control behavior time traces of both inexperienced (at the start of training) and experienced (at the end of training) participants.

Additional training data can be generated for the machine learning algorithm, this is known as *data augmentation*. Having more training data reduces overfitting and increases classification accuracy. Data augmentation can be achieved by transforming existing data (*data warping*), or by generating synthetic data (*oversampling*).

Oversampling may be achieved by utilizing a quasi-linear pilot model to simulate additional pilot behavior time traces. The proposed pilot model contains linear pilot-describing transfer functions and a remnant that represents the nonlinear contribution to human manual control behavior.

3

Artificial Neural Networks

As computing power increases, previously virtually impracticable tasks become executable by artificial neural networks. With their rise in popularity more and more of the potential of artificial neural networks is exploited. Applications span nearly every field of industry and academia; from translating text to recognizing images or even generating completely new images from scratch.

This chapter will first explain the working principle behind this popular type of machine learning algorithm. Next, different types of artificial neural networks are presented and their applicability for time series classification is discussed.

3.1. Traditional Artificial Neural Networks

To begin to explain the working principle of artificial neural networks (ANN), it is easiest to start with the simplest form: feedforward neural networks, also known as multilayer perceptrons (MLPs). This type of network is often referred to as the 'traditional ANN'. The concept's earliest description stems from the 1800s, but the first realisation is described by Mcculloch and Pitts (1943). Due to the lack of computational resources ANNs found no practical application until the late 1980s.

The goal of a feedforward neural network is to approximate some function f^*. For example if there is a classifier that matches some input $\mathbf{x}$ to a class y (i.e. $y = f^*(\mathbf{x})$), then a feedforward network tries to replicate this mapping with a function f with parameters $\boldsymbol{\theta}$ (i.e. $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$). The values for $\boldsymbol{\theta}$ are attained by learning from example mappings between input $\mathbf{x}$ and class y (Goodfellow, Bengio, & Courville, 2016).

3.1.1. Example of Simple Feedforward Neural Network

To better understand the description above, a schematic depiction of a fully connected feedforward neural network is provided in Figure 3.1. The circles in Figure 3.1 indicate neurons, the colored rectangles indicate layers. Each arrow between neurons is a connection, a 'fully connected network' means that each neuron is connected with every neuron on the next layer. The example feedforward neural network provided in Figure 3.1 has an input layer with two neurons, one hidden layer with three neurons, and an output layer with two neurons. Therefore, this specific network is able to take two input values, and produce two output values. For a classification task the amount of output neurons is equal to the amount of classes the network must be able to distinguish. Similarly, the amount of input neurons is equal to the amount of input variables with which the classification must be performed. So for example, the network depicted in Figure 3.1 could be 'trained' to distinguish cats and dogs (two outputs) based on weight and size (two inputs).

To explain how the neural network from Figure 3.1 could learn to distinguish two classes based on two inputs, it must first be described how data flows through the neurons of a feedforward neural network. Each neuron can hold a value, for input neurons this value is simply the provided input value, but for all other neurons this value is a function of previous neuron values. The value a neuron takes on is called the activation value (Nielsen, 2015).

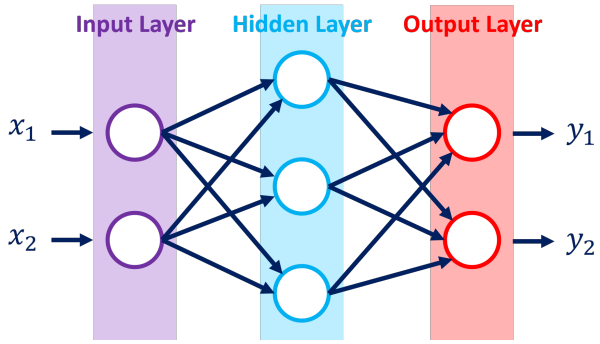


Figure 3.1: Schematic depiction of simple fully connected feedforward neural network with an input layer with two neurons, one hidden layer with three neurons, and an output layer with two neurons

An explanation of how these activation values are calculated will be provided using Figure 3.2. The activation values of the neurons in the hidden layer are denoted by a_i, each connection between neurons has an associated **weight** denoted by w_i, and each arriving neuron has an associated **bias** denoted by b_i. In Figure 3.2 focus is put on the first neuron of the hidden layer. The activation value of this neuron, a_1, can be calculated with equation Eq. (3.1).

$$a_1 = \phi\left(w_1 \cdot x_1 + w_2 \cdot x_2 + b_1\right) \tag{3.1}$$

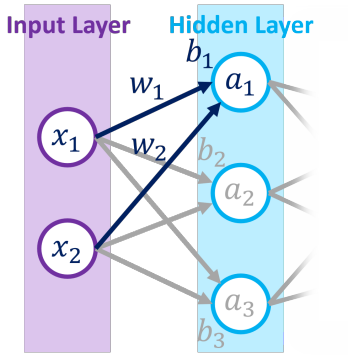


Figure 3.2: Example of values flowing from an input layer to a hidden layer. Each connection between neurons has an associated weight, and each arriving neuron has an associated bias

Here ϕ is the **activation function**. Activation functions are used to have more control over the activation value, by selection of activation functions certain properties can be introduced to the network to optimise its performance. For example, if a network is trained to replicate some non-linear behaviour then a non-linear activation function can be applied to introduce non-linearities into the network's output. An example of a popular non-linear activation function is the sigmoid function σ (Eq. (3.2)), this function squishes the activation value between zero and one. A similar popular activation function is the hyperbolic trigonometric tanh function (Eq. (3.3)). Unlike the sigmoid function the normalized range of tanh is minus one to one, giving it the advantages of being better at dealing with negative numbers (Patterson & Gibson, 2017).

In classification tasks the most commonly used activation function is the Rectified Linear Unit (ReLU) (Eq. (3.4)). The use of this function as an activation function for classification tasks was introduced by Nair and Hinton (2010). Networks with ReLU activation functions are more easily optimised than networks with more complex activation functions (e.g. the sigmoid function) (Ramachandran, Zoph, & Le, 2017). (Graphs of functions Eq. (3.2), Eq. (3.3), and Eq. (3.4) are provided in Appendix A)

$$\sigma\left(x\right) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

$$\tanh\left(x\right) = \sinh\left(x\right) / \cosh\left(x\right) \tag{3.3}$$

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases} \tag{3.4}$$

The activation values of all neurons in the hidden layer are calculated in the same way that a_1 was determined, i.e. by taking the sum of the connected input neurons multiplied by their respective weight and adding a bias before applying an activation function, this is shown in Eq. (3.5). Here $\boldsymbol{a}$ is vector with the activation values of hidden layer, $\boldsymbol{W}$ is a matrix containing all weights between the input layer and the hidden layer, $\boldsymbol{x}$ is a vector with the input values, and $\boldsymbol{b}$ is a vector with the biases of each neuron in the hidden layer.

$$\boldsymbol{a} = \phi\left(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}\right) \tag{3.5}$$

Once the activation values of the hidden neurons are determined they can pass on their values to the output layer. The values of the output neurons are determined almost completely the same the values of the hidden neurons. For example Eq. (3.6) and Figure 3.3 show how the value of the first output neuron, y_1, is calculated. (Mind that the weights and bias shown in Figure 3.3 are *not* the same weights and bias as shown in Figure 3.2, they only have the same subscript to simplify the notation of this example). This can be extended to calculate the vector with all output values $\boldsymbol{y}$ as shown in Eq. (3.7), now $\boldsymbol{W}$ is a matrix containing all weights between the hidden layer and the output layer, and $\boldsymbol{b}$ is vector with the biases of the output neurons.

$$y_1 = \phi\left(w_1 \cdot a_1 + w_2 \cdot a_2 + w_3 \cdot a_3 + b1\right) \tag{3.6}$$

$$\boldsymbol{y} = \phi\left(\boldsymbol{W} \cdot \boldsymbol{a} + \boldsymbol{b}\right) \tag{3.7}$$

A difference between the calculation of the output neurons activation values and the calculation of the hidden neurons activation values, is that the output neurons will often have a different activation function. In network designed for a classification task, such as the example network in this section, the output neurons will most commonly apply a **softmax** activation function. The softmax function returns the probability distribution over mutually exclusive output classes (Patterson & Gibson, 2017). This can be explained more easily by coming back to the cats and dogs classification example: if output neuron one (y_1) indicates cats and output neuron two (y_2) indicates dogs, then a network that is fully confident it is classifying a dog would have output $\boldsymbol{y} = [0, 1]^T$. Now if the network was not as sure (perhaps it is classifying a Chihuahua) it could have an output like $\boldsymbol{y} = [0.4, 0.6]^T$, indicating that it is 60% certain it is a dog. If the output layer has n neurons, then a softmax function can be applied to the output vector $\boldsymbol{y}$ using Eq. (3.8). After applying the softmax activation function the sum of the output neurons should be equal to one (total probability of 100%).

$$s\left(\boldsymbol{y}\right)_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{3.8}$$

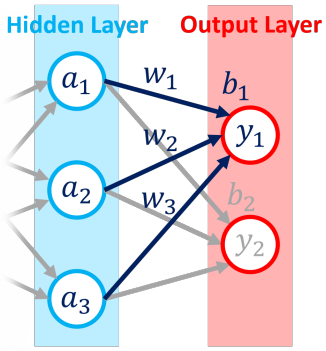


Figure 3.3: Example of values flowing from a hidden layer to an output layer. Each connection between neurons has an associated weight, and each arriving neuron has an associated bias

3.1.2. Training Artificial Neural Networks

Above description has shown how a feedforward neural network produces output values based on its input. It has been explained that the weights and biases (and activation functions) in the network combine and modify the input values into output values. But how does the network 'know' what weights and biases to use to produce the desired output values? This is found through **training**: the network 'learns' the best set of trainable parameters (weights and biases) to produce the desired output. But to find the 'best' set of parameters, there must first be a measure of how well the network is performing. This measure of performance is captured the **loss function**. Loss functions quantify the correctness between the predicted (network) output and the ground truth output (Patterson & Gibson, 2017). In other words: the loss function provides a number that quantifies how wrong the network output is, this number is also known as the loss.

The form of loss calculation described above (comparing model output to ground truth output) is only possible if there is correctly *labeled* data available. The form of machine learning where labeled data is used to train learning algorithms is known as *supervised learning*.

There are different loss functions that can be used to calculate the loss. The choice of loss function is completely dependent on the task that the network is trying to accomplish. A popular loss function for networks performing regression tasks that require a real valued output is the Mean Squared Error (MSE) loss (Patterson & Gibson, 2017). This loss function simply calculates the squared error between the network output and the ground truth output and averages this between all points in the dataset. The MSE loss function for a model with one output neuron is shown in Eq. (3.9), the same loss function for a model with multiple output neurons is given in Eq. (3.10). In these equations $\boldsymbol{W}$ is a matrix containing all the weights of the network, $\boldsymbol{b}$ is a matrix containing all the biases of the network, N is the scalar amount of points in the dataset, $\hat{y}$ is the network output, y is the ground truth output, and M is the amount of output neurons. The MSE loss function as given by Eq. (3.9) very clearly demonstrates the definition of loss: the difference between network output and actual output.

$$L\left(\boldsymbol{W},\boldsymbol{b}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2 \tag{3.9}$$

$$L\left(\boldsymbol{W},\boldsymbol{b}\right) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\hat{y}_{ij} - y_{ij}\right)^2 \tag{3.10}$$

As previously mentioned the choice of loss function is dependent on the task. The example feedforward neural network that has been discussed throughout this section is designed to perform a classification task. This means that for a certain input the network outputs the probabilities of each class' chance of belonging to the input. The network can then simply select the class with the highest probability of being correct to classify the input. When the actual probabilities output by the network are of greater interest than the hard classification **logistic loss functions** are used (Patterson & Gibson, 2017). These loss functions optimize for the maximum likelihood estimation (i.e. predicting the most probable probability of each class). A loss function that accomplishes this for a mutually exclusive classification task with M classes is the *categorical cross-entropy loss function*, given by Eq. (3.11).

$$L\left(\boldsymbol{W},\boldsymbol{b}\right) = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\cdot\log\hat{y}_{ij} \tag{3.11}$$

Now that a measure of performance of the network (loss) has been established, it can be described how the network 'learns' from data. This learning, or training, of the network is done by an algorithm called **backpropagation**. The goal of backpropagation is to compute the partial derivatives $\partial L/\partial w$ and $\partial L/\partial b$ of the loss function with respect to any weight w and bias b in the network (Nielsen, 2015). The mathematical description of how exactly this algorithm operates is extensive (such a description is provided by Nielsen (2015)) and out of scope of the explanation provided in this section. Therefore, a brief approximate description will be provided here instead.

As the name suggests backpropagation moves backwards through the network, starting at the output layer. At the output layer the loss is calculated and it is determined how the output neurons values

$\hat{\boldsymbol{y}}$ should change to reduce this loss (each individual output neuron value must either go up or down and some by a greater magnitude than others). Next, it is determined how the weights $\boldsymbol{W}$, biases $\boldsymbol{b}$, and activation values $\boldsymbol{a}$ of the previous layer should change to obtain the desired changes to $\hat{\boldsymbol{y}}$ (remember that the network output is calculated as shown in Eq. (3.7)). The desired change in activation values $\boldsymbol{a}$ are again a function of different weights, biases, and activation values of the layer before that. Again it is calculated how these weights, biases, and activation values must change. By repeating this process and propagating back through the network there will eventually be mapping of how each weight and bias in the network should proportionally change to reduce the loss, i.e. all $\partial L/\partial w$ and $\partial L/\partial b$ are found. Once all the partial derivatives are known the weights and biases are updated in a proportion that most effectively reduces the loss. The magnitude of the change in weights and biases (along the found proportions) per learning step is determined by a user input scalar parameter known as the **learning rate** α. The magnitude of α has a big influence on the learning performance: if α is too small it can take many training steps for the network to approach minimum loss (and it will be more likely to get stuck in a poor local minimum), and if α is too large it can easily overshoot the local minima. A schematic depiction of how α influences the training performance is shown in Figure 3.4, here θ indicate the network trainable parameters and L is the loss.

The learning algorithm (**optimizer**) explained above is known as *Gradient Descent*. This optimization method can be very computationally expensive, especially for large networks, because it requires calculation of every single partial derivative before taking a step. Therefore other optimizers have been introduced that speed up the learning process. For example by only determining the partial derivatives of a stochastically selected subset of model parameters, or by using an adaptive learning rate α. Examples of optimizers that combine these types of techniques (and more) are *AdaGrad* (Duchi, Hazan, & Singer, 2011), *RMSProp* (Tieleman & Hinton, 2012), *AdaDelta* (Zeiler, 2012), and most popularly *Adam* (Kingma & Ba, 2014).

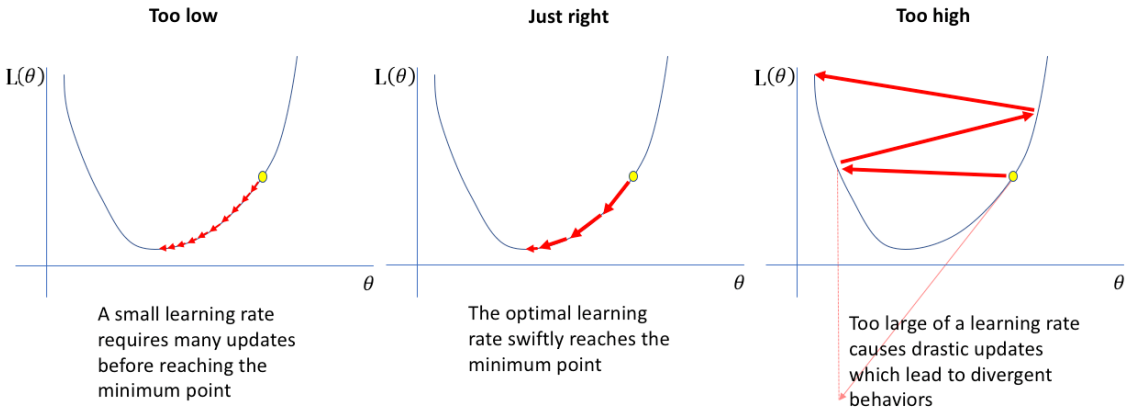


Figure 3.4: Influence of learning rate α on loss L minimization. Image taken from https://www.jeremyjordan.me/nn-learning-rate/

3.1.3. Improving Artificial Neural Networks' Performance

Now that the working principle of artificial neural networks has been explained and a foundation has been laid on how these algorithms are able to learn from examples, it must be explained how the learning performance can be measured and improved. This section will briefly touch upon some basic principles that are essential to understand how network effectiveness can be enhanced.

As has previously been explained, ANNs require *training* data to learn. However, in order to evaluate the performance of ANNS, an additional data set is required: *testing* data. This additional data set is required to simulate the effectiveness of the trained neural network in a real application, i.e. it must be able to classify data that it has not seen yet. Therefore, the achieved performance on testing data is what indicates the potency of the ANN, *not* the performance on the training data. In order to obtain this additional data set, the complete data set is simply split into two parts: *train data* and *test data*.

To get accurate network performance measures, it should be ensured that there is an even distribution of classes in the train- and test data. Using the cat and dog example: if all the dog data points

end up in the training data and all the cat data points in the testing data, then the network only learns to recognize dogs, but is evaluated in its ability to recognize cats.

During the training of the network, the entirety of the training data can be passed to the network multiple times. Each cycle the backpropagation algorithms makes through the training data is called an *epoch*. The first few epochs are beneficial to increase the classification performance of the network, since the backpropagation algorithm keeps updating the weights to better fit the training data. However, after too many epochs, the neural network becomes specifically good at recognizing the training data, but its performance starts deteriorating on the test data. This is known as *overfitting* (Nielsen, 2015). A schematic depiction of what this looks like is shown in Figure 3.5.

Overfitting can be recognized by increasing evaluation loss, or decreasing evaluation accuracy. During training the trainable network parameters should be saved in *checkpoints*, so that the best parameters (right before overfitting) can be retrieved after the training is done.

Besides the trainable model parameters, there are also fixed parameters that can be tuned to make networks train better and faster. These tuning parameters are known as **hyperparameters**. *"Hyperparameter selection focuses on ensuring that the model neither underfits nor overfits the training dataset, while learning the structure of the data as quickly as possible"* (Patterson & Gibson, 2017). Examples of hyperparameters are the amount of hidden layers, amount of hidden neurons, or the learning rate.

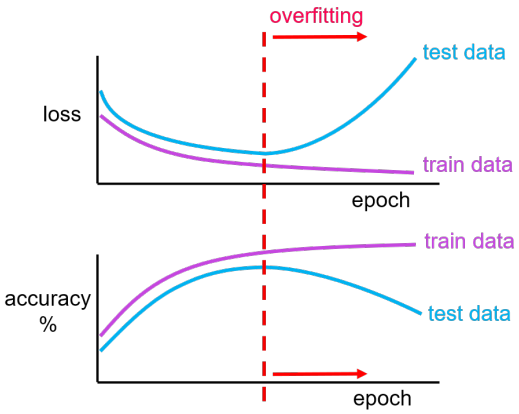


Figure 3.5: Schematic depiction of what *overfitting* looks like.

The previous sections provided an explanatory example of a simple feedforward neural network, also referred to as 'traditional ANNs'. In practice, these networks will have more hidden layers and more neurons per layer than was depicted in Figure 3.1, but the working principle remains the same. If a network has a large number of neurons (and layers) it is called a deep network. The next section will discuss different and more complex types of deep networks that have been developed to tailor to specific functionalities.

3.2. The Four Major Architectures of Deep Networks

There is a very wide range of existing forms of artificial neural networks (ANN), but each form excels at a different type of task. These different 'forms' of ANNs are distinguished by their 'architecture'. The architecture of a neural network describes the way the neurons operate and how they are connected. In this section an overview will be provided of different types of neural networks and their respective advantages.

To provide a list of *all* existing artificial neural network architectures is unworkable, as new architectures are constantly being developed, the list would never be complete. However, these endless examples of artificial neural network architectures being developed do generally rely on the same principles. Often these principles are combined in sequence or parallel to produce new architectures, which fundamentally exist out of the same building blocks. A division of 'the four major architectures of deep networks' is described by Patterson and Gibson (2017) (here 'deep networks' are defined as neural networks with a large number of parameters and layers). The four major network architectures are as follows:

- Unsupervised Pretrained Networks (UPNs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Recursive Neural Networks (Recursive NNs)

Extensive descriptions of each of these architectures are provided by Patterson and Gibson (2017). In this report, however, each of these fundamental architectures will only be briefly explained. This is because not all four of these architectures are designed to be directly applicable for time series classification (TSC). Still, many state of the art TSC ANNs combine components of aforementioned architectures to create newer and better performing neural networks. Therefore this elementary understanding of the major types of neural network structures will be crucial to comprehend the neural network architectures that will be proposed for this research. More detailed and specific descriptions of those proposed ANNs will be provided in Section 3.3.

3.2.1. Unsupervised Pretrained Networks

As the name suggests, Unsupervised Pretrained Networks (UPNs) undergo unsupervised learning before they start (supervised) training with labeled data. More specifically, there is a greedy layer-wise *unsupervised* pre-training phase followed by *supervised* fine-tuning (Erhan, Courville, Bengio, & Vincent, 2010). Simply put, the neural network is first accustomed to the training data before it starts the training phase.

The first examples of this method are Hinton, Osindero, and Teh (2006) who trained a Deep Belief Network (DBN), and Bengio, Lamblin, Popovici, and Larochelle (2006) and Ranzato, Poultney, Chopra, and LeCun (2007) who used stacked auto-encoders (both these networks are examples of generative models, they are described in more detail in Patterson and Gibson (2017)). These generative models are tasked to generate output data that resembles the input data. The idea is that, by replicating the input data, the generative models can already learn low-level features from data without requiring labels. Using the cat and dog image classification analogy: it is as though the neural network already learns some of the features of cats and dogs, before learning which feature belongs to which animal. This way the network already knows what to look for when it starts its supervised learning process. By doing so, the weights of the network are initialized such that they are closer to a global optima when entering the supervised training phase (this avoids the network from getting stuck in a poor local minimum (Bengio et al., 2006)).

A visual representation of above described method is shown in Figure 3.6 (Wang & Gupta, 2015). Here a Convolutional Neural Network (explained in the next subsection) is tasked to detect objects in video images. By first performing unsupervised pretraining the Convolutional network already shapes its weights (called filters) and later fine-tunes them during the supervised learning phase. Observe how the filters already start to form the general shapes during unsupervised pre-training which are later more prominently defined during fine-tuning. If there was no pre-training then all the filters in Figure 3.6 (a) would be randomly initialised and look like noise. Research has shown that unsupervised pre-training

often leads to lower classification errors than random initialization, especially when the learning data is sparse (Erhan et al., 2010).

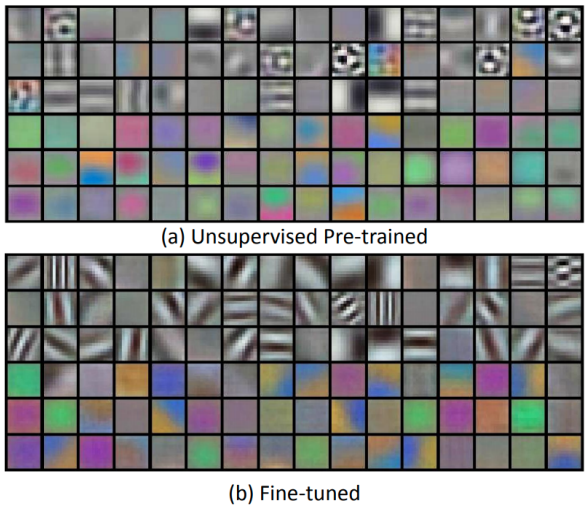


Figure 3.6: Convolutional filters visualisation. (a) The filters of a convolutional layer after unsupervised pretraining. (b) The same convolutional layer after supervised training (fine-tuning). Image taken from Wang and Gupta (2015).

The research of Wang and Gupta (2015) is an example of the previously made remark that the four major architectures of deep networks are often combined (in this case an Unsupervised Pretrained Network is combined with a Convolutional Neural Network). The next subsection will elaborate on working principle of Convolutional Neural Networks, and explain how the 'filters' shown in Figure 3.6 work.

3.2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) emerged due to the need of effective feature extraction from images. Traditional forms of ANNs tend to struggle with the computational complexity required to handle image data (O'Shea & Nash, 2015). Simple images of a small size, such as the MNIST database (LeCun & Cortes, 2010) (a large labeled database of 28x28 pixels images of handwritten digits), can be effectively classified by traditional ANNs. However, once the size of the image increases and the objects in the images become more complex these traditional ANNs fall short. The idea behind CNNs (as first introduced by LeCun, Bottou, Bengio, and Haffner (1998)) is that not every pixel is used as a separate input to a deep network (and fed through many layers with the hope of generalizing), but instead higher order features are extracted from the data which will be more likely to generalize between different data sets.

The biological inspiration for CNNs is the visual cortex in animals (Eickenberg, Gramfort, Varoquaux, & Thirion, 2017). The cells in the visual cortex each cover a subset of the visual input (visual field) of the eye. This way the brain is able to recognize shapes and edge-like patterns.

Conceptually, if a computer was tasked to recognize a beach, a traditional ANN would look at every grain of sand individually, whereas a CNN would instead look at the shape of the coast line and the color of the sand.

There are two reasons why the traditional ANN does not perform well at image classification: one, this granular approach does not generalize well with other instances, and two, it takes way too much memory and computational power (imagine inspecting a 600x600 pixels gray scale image pixel by pixel, this would already require 360,000 input neurons, i.e. 360,000 weights for each neuron in the first hidden layer and this would be three times more for a RGB (Red Green Blue) color image).

CNNs handle image data more efficiently by applying multi-dimensional sliding filters to extract features from the image (this will be explained in more detail in the next paragraph). In Figure 3.7 a schematic depiction of a CNN trained to distinguish a city from a beach is given. Notice how the input data is reduced to smaller arrays until it is ultimately fed to two neurons that decide the class of the image. There are five layers visible in Figure 3.7: the input layer, the convolutional layer, the pooling

layer, and two fully-connected layers (from left to right). Each of these types of layers will be briefly explained separately.

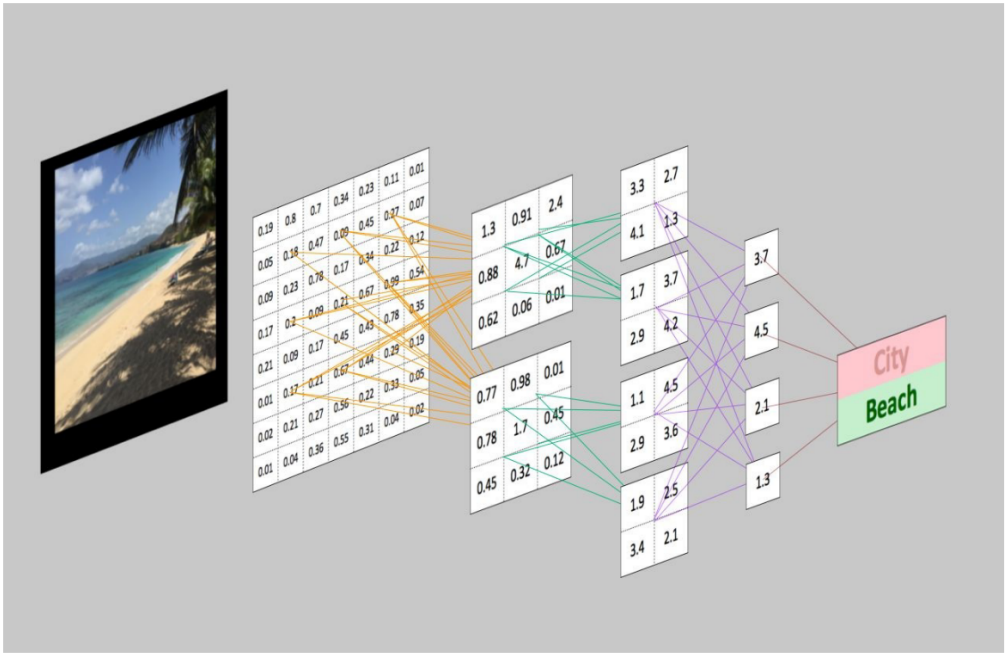


Figure 3.7: Schematic depiction of a CNN performing an image classification task. Image taken from Patterson and Gibson (2017)

The input layer is where the raw image data enters the network. If a gray scale image is used as input, then this layer has a two-dimensional shape: the width and height of the image (in pixels). Each value in this array represents the brightness of the pixel. If the input image has colors then the input layer has a three-dimensional shape: the width, height, and number of channels. Typically the number of channels will be three, for RGB images. For the sake of simplicity, the following explanation will assume a two-dimensional input.

The convolutional layer is what truly distinguishes the CNN from traditional ANN. In this layer there are multiple kernels (or filters). The user defines the amount of kernels and the size of each kernel. These kernels then slide across the input data to produce the convoluted feature (output data) (Patterson & Gibson, 2017). The kernel is multiplied with the input data creating a single entry in the output data for each step it takes sliding across the input data. An example of this process is shown in Figure 3.8. The size of the output data depends on the size of the kernel and the magnitude of the steps (also known as stride) it takes as it glides over the input data. The kernel in the example given in Figure 3.8 has values of one across the diagonals and zeros elsewhere, this forms a cross (a feature). By performing the convolution operation over the input data this kernel acts as a feature detector: the higher the value in the output data, the more present the feature (in this case a cross) is in that area. If, for example, the kernel only had ones on the diagonal then it would detect slanted lines. The values in the kernel are randomly initialized and then updated by learning (just like a regular ANN learns by updating its weights), this means that the CNN automatically learns what features to look for when training.

The output of the convolutional layer is also referred to as an activation map: it shows to what extend activated neurons decided to let information flow through (Patterson & Gibson, 2017). The activation map is commonly combined with an activation function (typically ReLu) to produce the final output of the convolutional layer.

There often are multiple convolutional layers present in a CNN, the first ConvLayer will recognize low-level features such as edges and colors, the following ConvLayer will combine these low-level features into higher level features, and so on. An example of these different filters at different convolutional layers is shown in Figure 3.9.

The pooling layer takes the output of the convolutional layer and decreases the spatial dimensionality, this is done to further reduce the amount of parameters and the computational complexity of the model (O'Shea & Nash, 2015). Pooling is a fairly simple mathematical operation where a sliding window glides over the input and produces a smaller output. The most commonly used pooling type is max-pooling (O'Shea & Nash, 2015), here each element in the output is the maximum value of the input in the sliding window. Much like in a convolutional layer, the user will have to define the window size (kernel size) and the sliding step magnitude (stride) of the pooling layer. An example of a max-pooling operation with a 2x2 kernel size and a stride of 2 is shown in Figure 3.10. Other types of pooling operations include normalizing or averaging the input of the sliding window.

The fully-connected layers at the end of the network enable the computation of class scores (Patterson & Gibson, 2017). The amount of output neurons of the fully-connected layers is equal to the amount of classes the network must be able to identify. Each pixel value that comes out of the pooling layer is a separate input to the fully-connected layers. This part of the network operates like a traditional ANN, the only difference is that the input has been simplified in such a way that generalisation becomes possible.

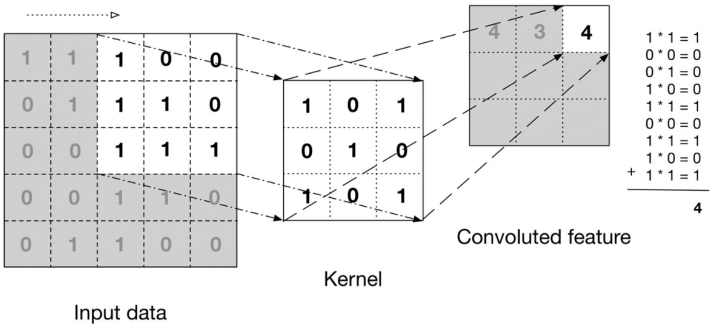


Figure 3.8: Example of kernel sliding across input data to produce convoluted feature (output data). Image taken from Patterson and Gibson (2017)

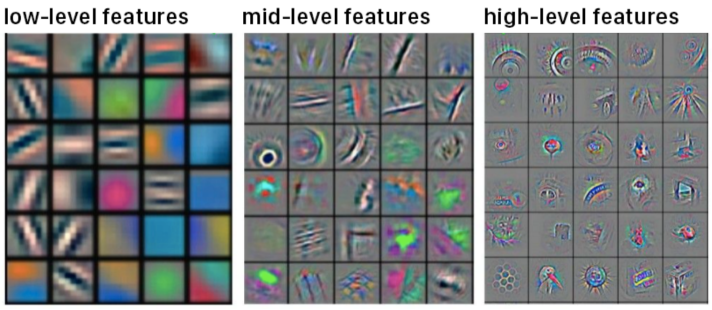


Figure 3.9: Examples of kernels at different convolutional layers. The first layer (on the left) has kernels that recognize low-level features, the next convolutional layer combines these low-level features into higher order features, and so on. Image taken from https://twopointseven.github.io/2017-10-29/cnn/

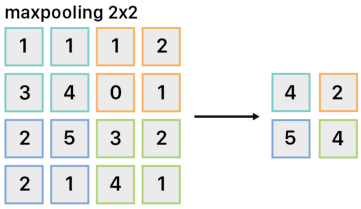


Figure 3.10: Example of max-pooling operation with a 2x2 kernel and stride of 2. The input is shown on the left, the output on the right. Image taken from https://twopointseven.github.io/2017-10-29/cnn/

Above explanation described the simplified working principle of each type of layer in a CNN. In practice CNNs will often stack multiple convolutional layers and pooling layers to gradually extract higher level features from the input image and ultimately classify them. In Figure 3.11 a schematic of a general CNN architecture is shown (in this image the ReLU activation function is indicated as a separate layer, but it can also be included within the convolutional layer). The weights of the convolution layers, as well as the fully connected layers, are calculated by training with a backpropagation algorithm.

Although CNNs were originally designed for computer vision applications, they are now also being applied to different types of tasks such as natural language processing (Wang & Gang, 2018) and also time series classification (Zheng, Liu, Chen, Ge, & Zhao, 2014).

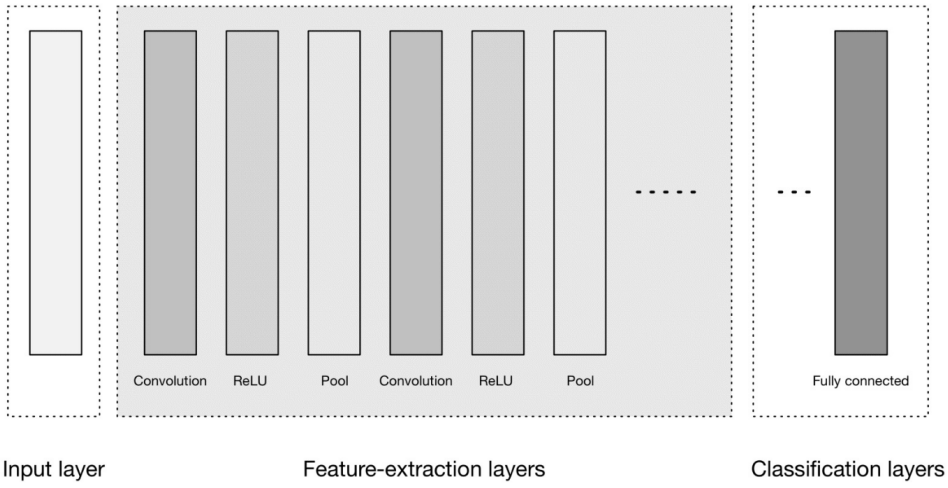


Figure 3.11: General CNN architecture schematic. Image taken from Patterson and Gibson (2017)

3.2.3. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are similar to feedforward neural networks (like the example that was provided in Section 3.1). The big difference is, however, that RNNs have the ability to send information between different time steps (Patterson & Gibson, 2017). This makes RNNs suitable for modeling functions that have an input and/or output that is time dependent.

To explain this consider the simple feedforward network that was presented in Figure 3.1. This network had an input vector $\boldsymbol{x}$ and produced an output vector $\boldsymbol{y}$. Now imagine that the input and output of this network had time dependencies between values, i.e. the input and output vector would look like $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$. Then it would be ineffective to consider each sample $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ individually per time step ($\boldsymbol{x}_i = \boldsymbol{x}(t = i)$, $\boldsymbol{y}_i = \boldsymbol{y}(t = i)$), since this would completely ignore the time dependencies between these samples. To solve this, a recurrent property must be added to the hidden neurons in the network; the hidden neurons in the network must have a memory of sorts so that they can pass on information between different time steps. This idea of recurrence has already been around since Minsky and Papert (1969), but was first dubbed a recurrent network by Rumelhart, Hinton, and Williams (1986). A schematic representation of what above explanation looks like is shown in Figure 3.12. In this example a *sequence* of inputs $\boldsymbol{x}(t)$ and outputs $\boldsymbol{y}(t)$ of length T and time step size dt is passed through an RNN. Precisely this ability of an RNN to be able to handle sequences is what makes it unique.

There are several ways an RNN can handle a sequence. In Figure 3.12 the handling is many-to-many, this means that for the sequence of input vectors there is an (equally large) sequence of output vectors. This type of handling is desirable if there must be an output at every time step. If the goal is to produce one output vector for a sequence of input vectors (as is the case for time series classification) then it is desirable to have a many-to-one mapping. This latter type of handling is shown in Figure 3.13, here an input time sequence of length T with two variables ($x_1(t)$ and $x_2(t)$) produces a single output vector. This can, for example, be used to classify an entire sequence as one class. Common examples of RNN applications are text generation, natural language processing, and time-series analysis (Patterson & Gibson, 2017).

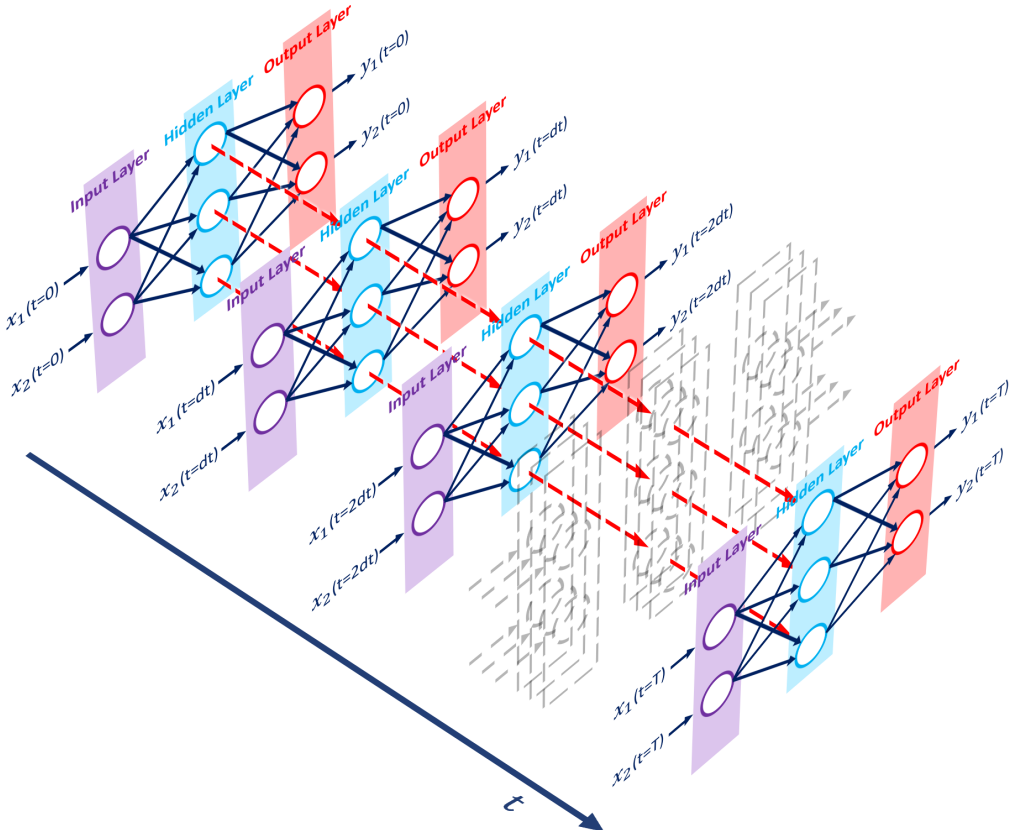


Figure 3.12: Schematic representation of an RNN passing information between different time steps (many-to-many mapping)

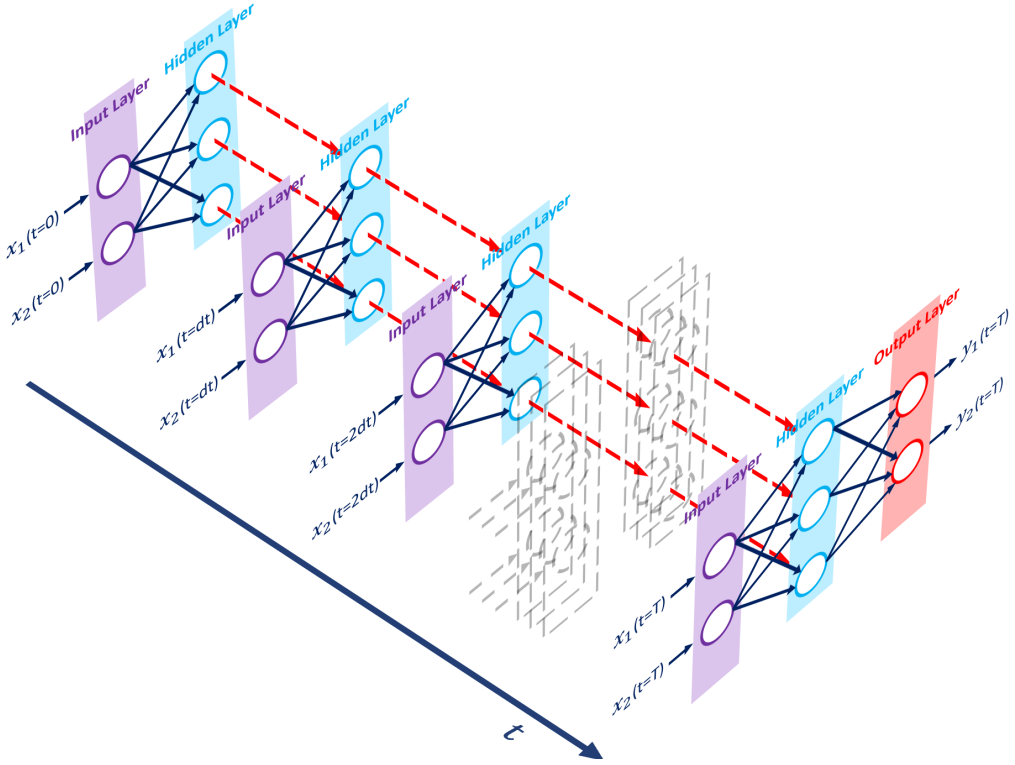


Figure 3.13: Schematic representation of an RNN passing information between different time steps (many-to-one mapping)

To enable the passage of information between different time steps, the hidden neurons in an RNN operate differently than hidden neurons in a regular feedforward network. Namely, hidden neurons in an RNN (also known as 'hidden units') have what is called a **hidden state** h. The hidden state of each hidden unit at t (h_t) is a function of both the input of that hidden unit at t (x_t) as well as the hidden state at the previous time step h_{t-1}. At each time step the hidden state that is transferred to the next time step is calculated with Eq. (3.12), the initial condition h_0 is equal to zero. Here ϕ an element-wise non-linear activation function, typically a sigmoid or hyperbolic tangent. The output of the hidden neuron at each time step z_t is calculated using Eq. (3.13) (Eq. (3.12) and Eq. (3.13) are taken from Donahue et al. (2014)). In these two equations W, U, and V are trainable weights and b indicate trainable biases. This inner working of an RNN hidden unit (or 'RNN cell') is schematically explained by Figure 3.14.

$$h_t = \phi\left(W \cdot x_t + U \cdot h_{t-1} + b_h\right) \tag{3.12}$$

$$z_t = \phi\left(V \cdot h_t + b_z\right) \tag{3.13}$$

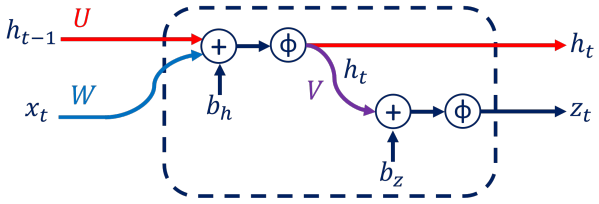


Figure 3.14: An RNN cell calculates a hidden state h_t and a cell output z_t based on the current cell input x_t and the cell's previous hidden state h_{t-1}

A major drawback of RNNs is that it is difficult to train them to learn long-term dynamics. A solution to this problem (also known as the 'vanishing gradient' problem) was found by Hochreiter and Schmidhuber (1997). Here a new type of recurrent neural networks, known as the **Long Short-Term Memory (LSTM)**, was introduced. The solution lies in the LSTM's hidden units' capability to learn when to forget previous hidden states and when to update hidden states given new information. This capability is achieved by the more complex design of hidden units in an LSTM, shown in Figure 3.15. These hidden units, also known as 'memory blocks' or 'LSTM cells', consist of four sub-units: the *forget gate*, *input gate*, *internal memory cell*, and *output gate*. A summarized description, as given by Versteeg (2019), of each of these sub-units is provided in the following paragraphs.

Forget gate: The forget gate determines how much of the previous internal cell state C_{t-1} will be kept. A 'remember' vector f_t^* is calculated based on the previous hidden state h_{t-1}, the current input vector x_t, their associated weights U_f and W_f, and a bias b_f, as is shown in Eq. (3.14) (at each gate the weight matrix belonging to h_{t-1} and x_t will be denoted as U and W, respectively. The bias of each gate is b).

A sigmoid function $\sigma(\cdot)$ squishes the values of f_t^* between 0 (=forget) and 1 (=keep). By taking the dot product (denoted by $\odot$) of C_{t-1} and f_t^* only a selective part of the previous cell state is remembered and carried over to the internal memory cell.

$$f_t^* = \sigma\left(U_f h_{t-1} + W_f x_t + b_f\right) \tag{3.14}$$

Input gate: The input gate produces a vector $\tilde{C}_t$ containing the potential values to update the previous internal cell state is calculated with Eq. (3.16).

Additionally a 'save' vector i_t (Eq. (3.15)) is computed based on the previous hidden state and current input. Through a dot product this save vector will determine which part of the potential cell state $\tilde{C}_t$ is transferred to the new cell state C_t

Notice the different activation functions used. The sigmoid function $\sigma(\cdot)$ used in Eq. (3.15) pushes the values of i_t between 0 and 1, allowing scaled selective conduction of $\tilde{C}_t$. On the other hand, the hyperbolic tangent function $\tanh(\cdot)$ used in Eq. (3.16) transforms the values of $\tilde{C}_t$ between -1 and 1, allowing both decrease and increase of the cell state.

$$i_t = \sigma\left(U_i h_{t-1} + W_i x_t + b_i\right) \tag{3.15}$$

$$\tilde{C}_t = \tanh\left(U_c h_{t-1} + W_c x_t + b_c\right) \tag{3.16}$$

Internal memory cell: As is shown in Eq. (3.17) the internal memory cell calculates the (new) internal cell state C_t by simple addition of the previously described components. Effectively, it is a combination of a selective part of the previous internal cell state C_{t-1} and a selective part of the potential cell state $\tilde{C}_t$.

$$C_t = f_t^* \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{3.17}$$

Output gate: The output gate determines what to output based on the (new) cell state C_t, the previous hidden state h_{t-1}, and the current input x_t (Eq. (3.18) and Eq. (3.19)). Notice that unlike in the RNN cell there is no separate cell output and hidden state. Instead the (new) hidden state h_t serves as both cell output to the next layer, as well as hidden state output to the next time step.

$$o_t = \sigma\left(U_o h_{t-1} + W_o x_t + b_o\right) \tag{3.18}$$

$$h_t = o_t \odot \tanh\left(C_t\right) \tag{3.19}$$

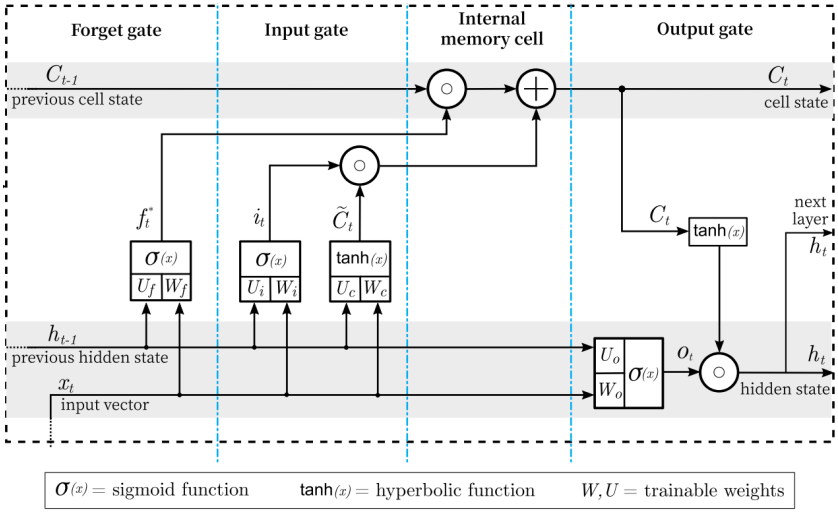


Figure 3.15: Schematic depiction of an individual LSTM cell's structure. Image taken from Versteeg (2019)

3.2.4. Recursive Neural Networks

The fourth and final major architecture that will be discussed is the *Recursive* Neural Network (Recursive NNs). These networks were first proposed by Sperduti and Starita (1997) with the goal of enabling neural networks to represent and classify structured patterns. By the introduction of a *generalized recursive neuron* networks gain the ability to model hierarchial structures in the training data set. In Figure 3.16 it is shown how different neuron models are suited for different types of input patterns. Technically, recurrent neurons are a specific form of recursive neurons that only considers the output of the unit in the previous time step, whereas generalized recursive neurons consider all the outputs of the unit that can be formed depending on the input.

More recent work by Socher, Lin, Ng, and Manning (2011) displays the effectiveness of recursive networks in parsing images and sentences. The idea of Recursive NNs, as they are applied in this work, is to recursively merge pairs of representations of smaller segments to get representations that cover bigger segments. This recursive pattern is followed until a representation of the entire input is made.

A practical example of how this hierarchical structure of representation of segments can be used is in image classification. It can be difficult to parse an entire image, especially if that image contains a

lot of different elements. Recursive networks allow a granular approach that merges semantic representations of segments to produce a classification for the entire scene. An example of what this looks like is shown in Figure 3.17, from each segment features are extracted and fed to a deep learning network that produces a semantic representation. These representations are then recursively combined, as previously described. At each recursive step that combines two semantic representations, the Recursive NN produces three outputs (Socher et al., 2011): 1) a score that is higher when neighboring regions should be merged into one region, 2) a new semantic representation of this merged region, and 3) the class label of the merged region. Following this recursive pattern will eventually provide a class label of the entire scene.

The same algorithm can also be used to parse natural language sentences (Socher et al., 2011). In precisely the same granular way that the image was classified, a sentence can be parsed. Fist words are mapped to semantic representations, which are then merged into phrases, which are also mapped to a semantic representation and merged. The recursive neural network produces the same three outputs that were previously described, the only difference is that the class labels are now phrase types such as 'noun phrase' or 'verb phrase'. Again, following the recursive pattern a semantic representation of the entire sentence can be composed.

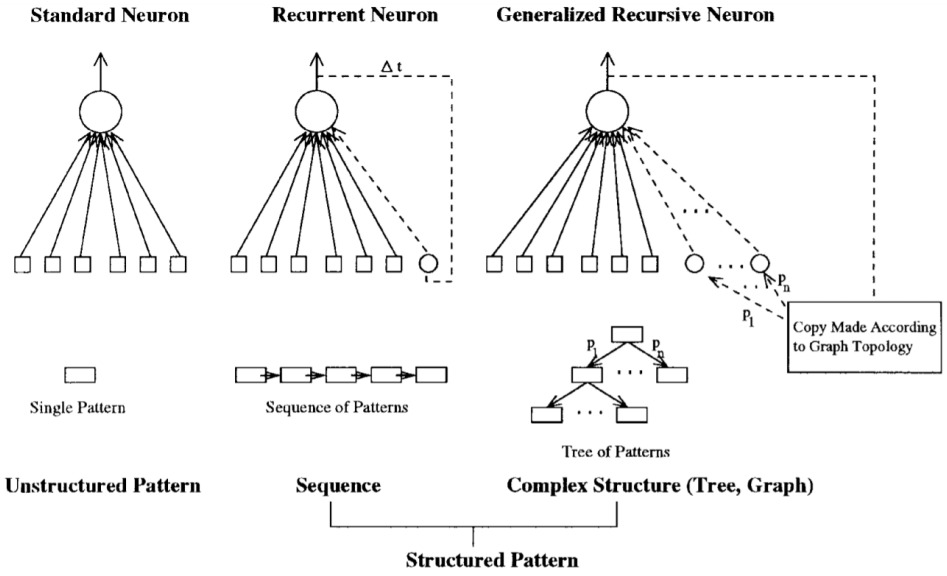


Figure 3.16: Different neuron models are suited for different inputs. The standard neuron is good for handling unstructured patterns, the recurrent neuron for sequences of patterns, and the recursive neuron for (hierarchy) structured patterns. Image taken from Sperduti and Starita (1997)

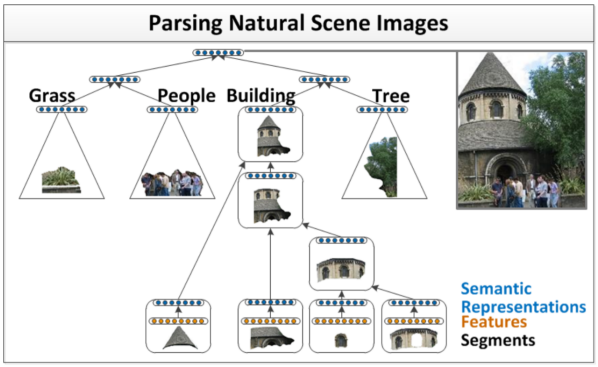


Figure 3.17: Illustration of recursive neural network architecture which parses images. Segment features (orange) are first mapped into semantic representations (blue) and then recursively merged by the same neural network until they represent the entire image. Image taken from Socher, Lin, Ng, and Manning (2011)

3.2.5. Deep Network Architectures Summary

Table 3.1 summarizes the information that has been provided about the four major architectures of neural networks: Unsupervised Pretrained Networks, Convolutional Neural Networks, Recurrent Neural Networks, and Recursive Neural Networks (Patterson & Gibson, 2017). A short explanation of each of the four major architectures is provided under the 'description' column of Table 3.1.

The 'purpose' column gives examples of types of tasks that each class of networks is typically used for. Note that the architectures are not limited to the listed purposes and that they are also applied outside of the domain that they were designed for (e.g. Zhu, Chen, and Peng (2019) deploy a CNN architecture to estimate the remaining useful life of bearings based on time frequency representations of degradation signals, or Mohan and Gaitonde (2018) uses an RNN to model reduced order temporal dynamics of turbulent flow).

Table 3.1: Summary of the information that has been provided about the four major architectures of neural networks (Patterson & Gibson, 2017).

Architecture	Description	Purpose
Unsupervised Pretrained Networks (UPNs)	These networks undergo an unsupervised pretraining phase in which they are tasked to reconstruct input data. This generation of data can be the goal itself, but it can also be utilized to initialize the trainable parameters of the network closer to a global optimum.	Reconstruction of the input data can be used to • Initialize weights (to improve learning performance, especially if training data is sparse) • Generate images, sounds, or video
Convolutional Neural Networks (CNNs)	This architecture of neural networks was built to effectively handle multi-dimensional data. It has an unmatched performance in analyzing visual imagery. A sliding kernel, or filter, is applied to the input data to create a feature map. By extracting and combining features of different levels of abstraction, CNNs can recognize larger constructions of objects.	Designed for computer vision tasks such as • Image classification • Image processing (e.g. compute a steering command for an autonomous vehicle based on camera input) • Three-dimensional data processing (e.g. MRI data or 3D shape data)
Recurrent Neural Networks (RNNs)	The recurrent property of the neurons in this class of networks allows the modeling of time dependent behavior. By giving the neurons a hidden state (a memory of sorts), the network can incorporate data from previous time steps to produce output. This allows RNNs to 'understand' context in sequences with interdependencies.	Able to handle sequential data and thus applicable for e.g. • Natural Language Processing • Text generation • Sequence generation or classification
Recursive Neural Networks (Recursive NNs)	This class of networks recursively combines segments of input data to obtain a semantic representation of the entire input. This process produces a tree of patterns that enables the handling of complex structures.	Modeling the hierarchical structure of data can be used in • Natural Language Processing • Text semantic analysis • Image decomposition and classification

3.3. Artificial Neural Networks for Time Series Classification

This section will address which of the previously introduced types of Neural Networks may be applied for the TSC task of this thesis. As previously mentioned, the properties of the discussed major types of architectures are not mutually exclusive. This means that their strengths can be combined to optimize the TSC performance.

In order to identify what type of neural network would be applicable for this thesis, a review of existing ANN TSC literature is presented in Section 3.3.1. It was found that, from the four major architectures, mainly Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) are used for TSC. Recursive Neural Networks (Recursive NNs) are rarely mentioned in TSC literature. Unsupervised Pretrained Networks, however, remain a viable option to use in combination with other network architecture to enhance their performance.

After qualitatively comparing RNNs and CNNs, Section 3.3.2 will provide a more specific description of the selected ANN architectures for each stage of this thesis. Additionally, some potential hybrid architectures are presented.

3.3.1. Recurrent Neural Networks versus Convolutional Neural Networks

As has been stated in the introduction, the goal of this thesis is to effectively distinguish trained- and untrained pilots based on time traces of their control behavior. A similar thesis by Versteeg (2019) also used artificial neural networks to classify pilot behavior based on pilot time traces. However, the goal of Versteeg (2019) was not to recognize pilot skill level, but to identify the dynamics of the element the pilot was controlling. Although the goals of these theses are different, the definition of the task is identical: both theses attempt the utilize artificial intelligence to classify time traces of pilots. Therefore, the conclusions drawn by Versteeg (2019) are useful for the purpose of this thesis.

From the work of Versteeg (2019) it followed that LSTM networks (Hochreiter & Schmidhuber, 1997) can effectively be utilized to classify time recordings of pilot control behavior. In a preliminary experiment a comparison was made between the performance of an LSTM networks versus that of an SVM model(Evgeniou & Pontil, 2001). From this preliminary experiment it followed that the LSTM network was the favored option to perform the TSC task. Choosing LSTM (a form of Recurrent Neural Networks) is a logical option since these networks were specifically designed to handle sequential data (as has been discussed in Section 3.2.3). There are more examples of RNNs effectively analyzing human control behavior: for example Saleh et al. (2017) used an LSTM network to classify driving behavior, and Jain et al. (2016) used an LSTM architecture to anticipate driver activity.

Although RNNs appear to be the straightforward option for TSC, a literature review by Fawaz et al. (2019) found that Convolutional Neural Networks are the most widely applied architecture for the TSC problem. This is likely due to their robustness and their relatively short training time compared to RNNs (Fawaz et al., 2019). CNNs are quicker to train because they allow parallel computing of the trainable parameters, unlike RNNs which require sequential updates (due to the nature of the architecture). Another benefit of using CNNs is that it enables Class Activation Mapping (CAM), as first introduced by Zhou, Khosla, Lapedriza, Oliva, and Torralba (2016). CAM enables the visualisation of what part of the input data contributed the most for given a specific classification. This method was first applied for TSC by Wang et al. (2017) who highlighted what part of a univariate time series belonged to each class to make the model output interpretable. Model interpretability will be further discussed in Section 3.4.

It is difficult to predict which network architecture, RNN or CNN, will outperform the other. The performance of each network is largely dependent on the task, thus either architecture can excel in performance depending on the exercise. There is little literature available that makes a fair comparison between RNNs and CNNs. However, there are examples that show that CNNs can outperform RNNs in time series classification (Gao, Hendricks, Kuchenbecker, & Darrell, 2016) and time series modeling (Mittelman, 2015). Since no quantitative comparison can be made between the two networks, a qualitative trade-off was made by listing the advantages and disadvantages of each architecture. The result of this qualitative comparison is captured in Table 3.2.

Table 3.2: Advantages and disadvantages of using RNNs and CNNs for TSC.

Architecture	Advantages	Disadvantages
Recurrent Neural Networks	• Easy to implement since the network is specifically designed for sequential data • Proven effectiveness in classifying pilot control behavior (Versteeg, 2019) • Good at modeling time dependent data (Patterson & Gibson, 2017)	• May suffer from vanishing gradient problem (Bengio, Simard, & Frasconi, 1994) • Can not easily interpret network parameters • Takes more computation time to train network (Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019)
Convolutional Neural Networks	• Interpretable network parameters allow CAM (Zhou, Khosla, Lapedriza, Oliva, & Torralba, 2016) • Faster training of network due to increased parallelism (Bradbury, Merity, Xiong, & Socher, 2016) • Most commonly used for TSC (Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019)	• No proven effectiveness in the specific topic of this thesis • Not specifically tailored for sequential data handling, and thus potentially more difficult to implement

3.3.2. Motivation of selected methods

In this section, the collected information from literature will be utilized to choose which neural network architecture(s) will be used for this thesis. The previously provided explanations of the four major architectures described the overarching working principles of each class of neural networks. There are many options of different neural networks structures within each of these classes of architectures. Therefore, after selecting the overarching architecture, this section will also give a more detailed description of the exact neural network structure within its class that seems most promising for the purpose of this thesis.

There are two phases in this thesis: a preliminary phase (this report) and a main phase. The two phases both have different goals. Namely, the preliminary phase is to explore available methods from literature and to get a good understanding of the subject. In the preliminary phase there is also a preliminary experiment. This experiment is conducted to briefly test if, and how well, the collected theoretical method works in practice. This practical knowledge will also allow to compose a detailed plan of how time and effort should be spent in the main phase.

In the main phase of this thesis an experiment will be conducted with the aim to acquire new findings and potentially close an academic gap. The final conclusion of this thesis will be based on the results of the main phase. Therefore, in the main phase, an optimal neural network model must be extensively tested under different conditions to be able to gather legitimate conclusions.

In short, the two phases have different goals: the first phase is to explore and to test feasibility, whereas the second phase is to extensively experiment and optimize so that conclusive results can be found that add to the existing academic body of knowledge. Given the different nature of each phase, the decision was made to use different neural network architectures for the two phases. An explanation of what neural network structure will be tested in each phase (and why) is provided in the following paragraphs.

Preliminary phase: As has just been explained, the goal of this phase is to explore and to test feasibility. Therefore, it is important that the neural network architecture used for the preliminary experiment helps to fulfill this goal. Based on the information gathered in Table 3.2 it is was decided that Recurrent Neural Networks are the best suited for the preliminary experiment.

This decision was made based on the first two advantages listed in Table 3.2. 1) Since RNNs are

easy to implement for TSC, they require less time to employ. This is ideal for brief prototyping. 2) Given the proven effectiveness of RRNs to classify pilot control behavior (Versteeg, 2019), a lot of settings can be directly taken from the existing work. This allows to quickly test the feasibility of using deep neural networks to identify pilot skill level. If poor results are found, they are likely not because of bad settings, but because the concept is infeasible.

Since the decision to use RNNs for the preliminary experiment was largely based on the findings of Versteeg (2019), it was also decided to use the same neural network structure as was used by Versteeg. The neural network model structure used is a stacked LSTM structure inspired by Saleh et al. (2017), combined with the rule of thumb hyperparameter settings from Reimers and Gurevych (2017). An exact detailed description of this network is provided in Section 4.2.1.

Main phase: The main phase of this thesis aims to provide conclusive remarks about the usability and performance of deep learning for pilot skill level prediction. Based on the information shown in Table 3.2 it was decided that Convolutional Neural Networks are the class of neural networks that are most suitable to accomplish the goal of the main phase.

The three advantages of CNNs shown in Table 3.2 sum up the motivation to employ this architecture in the main phase. 1) The trained parameters of CNNs can be used to generate Class Activation Maps. This method visualizes the 'reasoning' of the network to predict a certain class as output. There are methods that produce these types of visualizations for RNNs too (as will be discussed in Section 3.4). However, it is favorable to be able to directly gain insight from the trained network parameters, since this mitigates the chances of misinterpreting model output and it removes the need of additional explainability algorithms. 2) Because of increased parallelism, CNNs require less training time than RNNs. This is specifically beneficial for the main phase of this research, since it will facilitate the large amount of training runs required for extensive testing and optimisation of the network. 3) Since CNNs are currently the most commonly used structure for TSC, it is an option that must be considered. Seeing that RNNs are already used for the preliminary phase of this thesis, investigating CNNs in the main phase will allow for a fair comparison between the performance of the two networks.

Although CNNs were designed to analyze images in computer vision tasks, there are numerous ways to employ them for time series classification. For example, in a research by Jiang and Yin (2015) the time signals were first converted to an image format, after which a CNN was utilized to classify those images. However, by selecting appropriate kernel sizes, CNNs can also be directly applied to time series data. In the case of a Univariate Time Series (UTS), for example, the data has only one dimension (time), unlike an image that has two dimensions (width and height). If the sliding filter of the convolution layer also only has one dimension, then it can still be used to produce convoluted features of the time trace. As a concrete example: if there is a filter of length three, with filter values $\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$, that is applied to an UTS, then convolution (multiplication) will produce a moving average of length three across the time series (Fawaz et al., 2019). When considering a Multivariate Time Series (MTS) as input to the convolutional layer, then the filter no longer has one dimension (time), but also has dimensions that are equal to the number of dimensions of the input MTS (Fawaz et al., 2019).

Wang et al. (2017) compared the classification performance of three deep neural network architectures to traditional TSC methods on 44 UTS. From this experiment it was found that Fully Convolutional Networks (FCN) and Residual Networks (ResNet) (both CNNs) can achieve comparable or better performance than state-of-the-art traditional TSC methods. The advantage of using deep neural networks is that, unlike the traditional TSC methods, they do not require any feature engineering or data preprocessing (Wang et al., 2017).

Fawaz et al. (2019) analyzed the TSC performance of nine different neural network architectures (out of which six were CNNs and one an RNN) on 85 UTS datasets and 12 MTS datasets. From this benchmarking research it was also concluded that FCN and ResNet achieve the best classification accuracy overall. Based on these findings, it was decided that precisely these two neural network models should be investigated during the main phase of this thesis.

Fully Convolutional Networks (FCN) have shown a proven effectiveness for semantic segmentation of images (Long et al., 2015). Wang et al. (2017) adopted the FCN as a feature extractor with its final output coming from a softmax layer. This FCN structure is shown in Figure 3.18, here BN stands for

Batch Normalization, this is applied to increase the convergence speed and improve generalisation (Wang et al., 2017). The numbers 128, 256, and 128 indicate the amount of filters in each convolutional layer. Notice that, unlike the general CNN architecture that was shown in Figure 3.11, there are no pooling operations between the layers. Instead, there is only a Global Average Pooling (GAP) operation over the feature maps in the classification layer, this makes the results easier to interpret and reduces the chances of overfitting (Lin, Chen, & Yan, 2014).

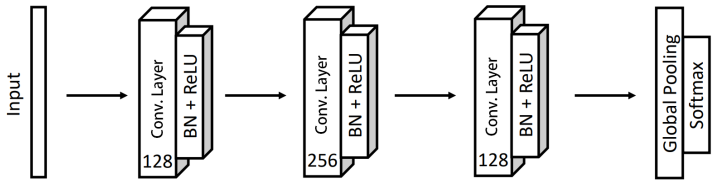


Figure 3.18: Network structure of the Fully Connected Convolutional Network. Image taken from (Wang, Yan, & Oates, 2017)

Residual Networks (ResNet) were introduced by He et al. (2015) as a framework to train deep neural networks for image recognition. Simonyan and Zisserman (2015) showed that adding more convolutional layers (thus making the network deeper) is beneficial for the accuracy in image classification. However, adding more layer also makes training of the network more difficult. He et al. (2015) developed a method that uses 'shortcut' residual connections to overcome these training difficulties, whilst still profiting the enhanced accuracy from greatly increased depth. Wang et al. (2017) adopted ResNet for TSC, the resulting structure is displayed in Figure 3.19. Again, BN is done after every convolutional layer and pooling operations are excluded until the GAP layer.

A newer (and potentially better) similar network architecture called *InceptionTime* was introduced by Fawaz et al. (2020). This network is similar in the sense that it also uses residual blocks and a GAP layer followed by a fully connected layer with softmax activation. However, there are *two* residual blocks, instead of *three*, and each of the residual blocks is now composed of three Inception modules (Szegedy et al., 2015), instead of the traditional fully convolutional layers. A major component of the Inception module is that it uses multiple sliding filters of different lengths that simultaneously extract features from the same time series.

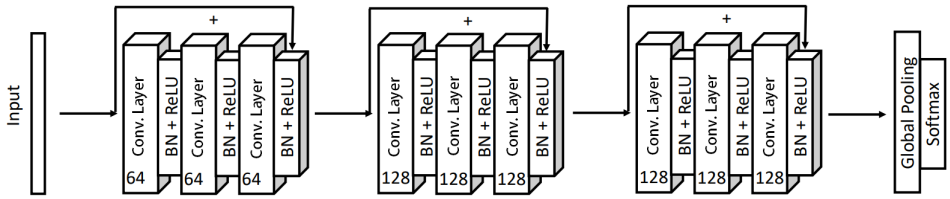


Figure 3.19: Network structure of the Residual Network. Image taken from (Wang, Yan, & Oates, 2017)

Lastly, it should also be mentioned that there are 'hybrid' architectures that could be interesting for the main phase of this thesis. For example, Ordóñez and Roggen (2016) designed a sequential combination of convolutional layers (for feature extraction) and recurrent layers (to model temporal dynamics). This structure outperformed baseline deep CNN in activity recognition. Yao, Zhang, Zhou, and Liu (2019) used a parallel structure to have both RNN and CNN simultaneously extract features from data, after which an attention mechanism selected the relevant features to ultimately classify breast cancer biopsy images. This research achieved state of the art classification results. Bradbury, Merity, Xiong, and Socher (2016) introduced Quasi-Recurrent Neural Networks, an architecture that uses convolutional layers in combination with a recurrent pooling function. These networks are up to sixteen times faster in train and test times than regular RNNs due to increased parallelism, whilst maintaining similar or better predictive accuracy.

Although these hybrid architectures appear promising, they are considered to be out of the scope of this thesis. Namely, the goal of this thesis is not to develop new complex neural networks to improve performance in an existing use case, but rather to test existing neural networks on a new use case.

Simpler proven networks will be more useful to test whether pilot skill level classification can be accurately done by ANN. That being said, if performance falls short, the aforementioned hybrid architectures may still be viable options.

3.4. Explainable Artificial Intelligence

This chapter has indicated the usefulness of artificial neural networks for many different purposes. However, it has not yet addressed one of the biggest challenges in artificial intelligence: the *black-box* problem (Castelvecchi, 2016). ANNs are considered black-box models because although they are excellent at mapping some input to some output, there is little visibility on the internal workings of the network models.

This section will highlight some of the recent developments in tackling the black-box problem to make artificial intelligence more transparent. Furthermore, specific explainability solutions for this thesis will be recommended.

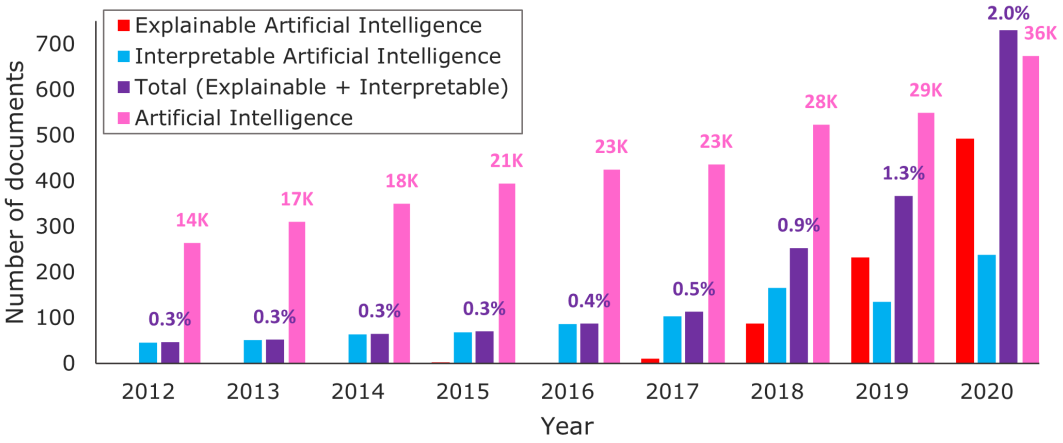


Figure 3.20: Number of documents found on Scopus per year with 'Artificial Intelligence' (pink), 'Interpretable Artificial Intelligence' (blue), and 'Explainable Artificial Intelligence' (red) in the Title, Abstract, or Keywords. The percentages express the total XAI documents (purple) compared to total AI documents (pink). Data retreived in April 2021.

Very early adaptation of artificial neural networks, consisting of only a few neurons, could still be relatively easily interpreted. However, over the last few decades deep learning models have been developed consisting of millions of parameters. Though these deep models have demonstrated empirical success in learning difficult complex (real-world) phenomena (as has been shown throughout this chapter), their increasing depth has only made them more opaque. This lack of transparency and interpretability is major drawback of these state of the art models (Došilović, Brčić, & Hlupić, 2018). Considering these shortcomings, eXplainable Artificial Intelligence (XAI) has become a surging topic of interest in the research community. In Figure 3.20 a quantitative indication of this rising popularity is expressed by the amount of work published regarding XAI per year in the last nine years. As a frame of reference, Figure 3.20 also indicates the total amount of work published regarding Artificial Intelligence in general, and what percentage of this was related to XAI. Evidently, the popularity of XAI rises faster than the popularity of AI in general, indicating a vast desire for more transparency in deep learning.

Arrieta et al. (2019) sums up three reasons why interpretability should be considered as an additional design driver to improve the implementation of Machine Learning (ML) algorithms:

- Interpretability helps to ensure impartial decision-making, i.e. it allows to detect (and correct) bias in the training dataset.
- Interpretability accommodates robustness by highlighting potential perturbations that could change the model output.
- Interpretability can act as an insurance that only relevant variables gather the found output, i.e. ensuring that there is a truthful relationship between input and output in the model reasoning.

This means that, in order to consider the explanation of a system useful, it should either 1) provide an understanding of the model mechanism and predictions, 2) visualize the model's discriminative rules for decision making, or 3) hint on what could perturb the model (Arrieta et al., 2019).

Different types of ML models require different explainability methods. Arrieta et al. (2019) conducted a literature analysis to construct a taxonomy capturing all the different categories of explainability methods that are being developed. In Figure 3.21 a summarized version of this taxonomy is presented. There are practically two classes of XAI in ML: *Transparent Models* and *Post-Hoc Explainability*. The former class encompasses models that are by themselves understandable, they require no post-hoc analysis to be interpretable (Arrieta et al., 2019). Deep learning models, however, fall into the secondly mentioned class of XAI. This means that these algorithms require post-hoc analysis to be made explainable. Two different types of post-hoc explainability will be discussed in the following paragraphs: *Model-Specific Explainability* and *Model-Agnostic Explainability*.

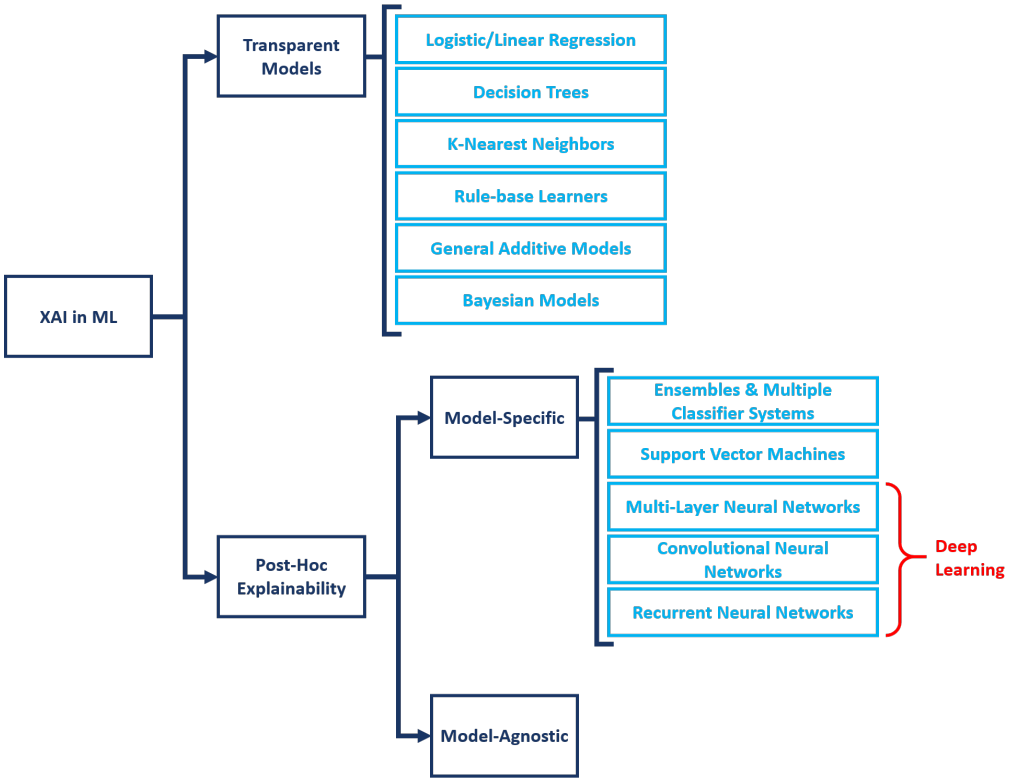


Figure 3.21: Taxonomy of explainability for different machine learning models. Based on findings from (Arrieta et al., 2019)

Model-Specific Post-Hoc Explainability: Model-specific explainability approaches exploit the unique characteristics of an ML model to make it interpretable. This means that these methods can only be applied to the ML model that they were designed for.

An example of a post-hoc model specific explainability method is the previously mentioned CAM, introduced by Zhou et al. (2016). This method requires a GAP operation to output the spatial average of each feature map of the last convolutional layer. The discriminative regions of the input that led to the model output can now be visualized by mapping back the predicted class score to the output of the GAP layer (Zhou et al., 2016). This procedure is depicted in Figure 3.22.

To illustrate the usefulness of this method, consider Figure 3.23. This figure shows a CAM of a trained FCN on the GunPoint dataset (Ratanamahatana & Keogh, 2005). This dataset contains recordings of hand motions of actors performing either of two actions: 1) the actor draws a gun from a holster and aims it at a target for approximately one second (class 1 = 'Gun'), or 2) the actor just sticks out their index finger and points at the target for one second (class 2 = 'Point'). Both the vertical and horizontal

displacement of the actors their hands were tracked, however, these appeared to be highly correlated (Ratanamahatana & Keogh, 2005). Therefore, the data in the archive only contains the horizontal displacement (making it an UTS). Fawaz et al. (2019) trained an FCN that had nearly 100% accuracy in disinguishing the two classes. Figure 3.23 shows how CAM can help to visualize how the trained FCN recognizes the two classes, based on discriminative regions in the data.

Far less contributions have been made for explaining RNNs. For comparison, the taxonomy of reviewed literature by Arrieta et al. (2019) only contains ten papers concerning explainability for RNNs, whereas there are 32 for CNNs. There are some visual explanation methods for RNNs (e.g. Arras, Montavon, Müller, and Samek (2017) and Karpathy, Johnson, and Fei-Fei (2015)). However they are not directly applicable for TSC, as these RNN visualization methods are developed for NLP.

Given the few contributions for explaining RNN (Arrieta et al., 2019) and the customization required to apply those existing methods to TSC, model-agnostic explainability appears to be the more promising candidate for the preliminary phase of this thesis.

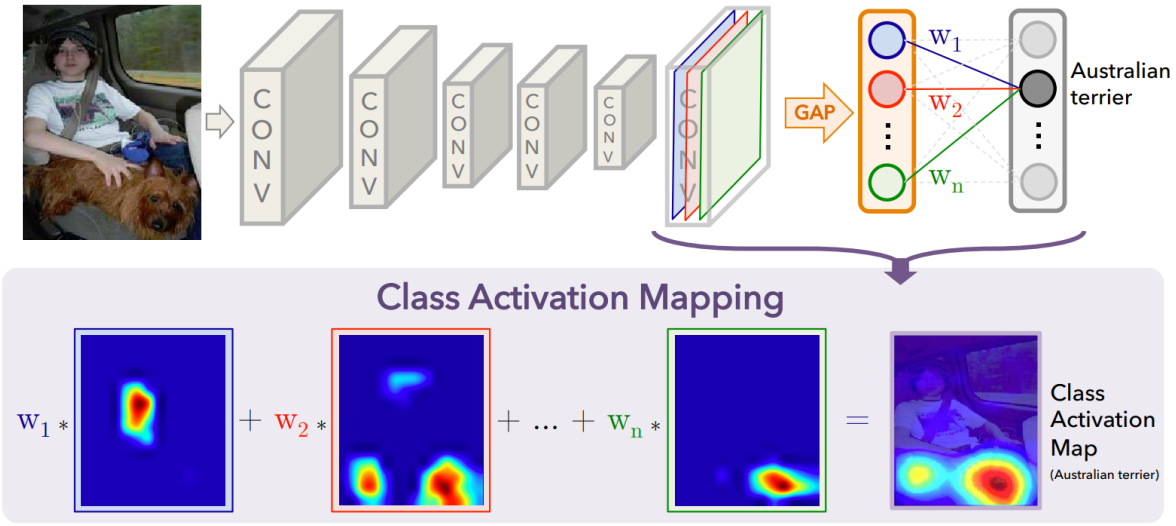
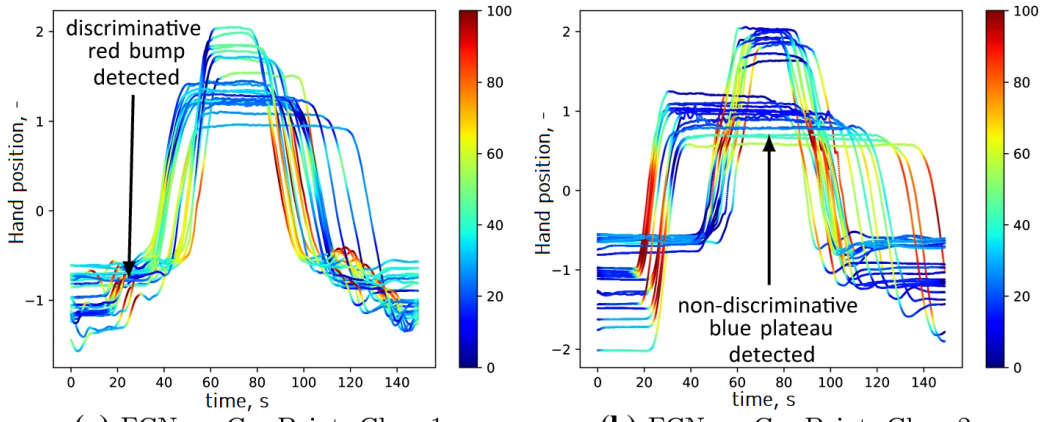


Figure 3.22: Class Activation Mapping explained: the predicted class score is traced back to the last convolutional layer to generate class-specific discriminative regions. Image taken from (Zhou, Khosla, Lapedriza, Oliva, & Torralba, 2016)



(a) FCN on GunPoint: Class-1 **(b)** FCN on GunPoint: Class-2

Figure 3.23: Example of Class Activation Mapping with FCN to highlight regions of the time series that contributed to class 1 (Gun) and class 2 (Point). Blue indicates regions with no contribution, red regions are area with maximum contribution. Image taken from (Fawaz et al. 2019).

Model-Agnostic Post Hoc Explainability: Model-agnostic techniques for post-hoc explainability are designed to be utilized on *any* model with the intent to extract some information of the inner workings of that model (Arrieta et al., 2019).

An example of such an explanation technique is Local Interpretable Model-agnostic Explanations (LIME), as introduced by Ribeiro, Singh, and Guestrin (2016). LIME explains predictions of any opaque classifier by simplification; it learns a linear model locally around the prediction that serves as an interpretable representation (Ribeiro et al., 2016). This method is only *locally faithful*, meaning that the provided explanation is only true for the instance of a prediction that is examined.

Another model-agnostic explainability technique is that of Lundberg and Lee (2017) called SHapley Additive exPlanations (SHAP). This method is a feature relevance explanation technique. SHAP uses a game theory inspired method to assign each feature (input variable) an importance value for a particular prediction (Lundberg & Lee, 2017). This method can be used for local explanation of single instances, but can also enable global explainability by aggregating local Shapley values (Hall, 2020).

To demonstrate the usefulness of SHAP, and to further explain the difference between local and global explainability, consider the following example. An artificial neural network is trained on a 'red wine quality' dataset (Cortez, Teixeira, Cerdeira, Almeida, Matos, & Reis, 2009) to predict the quality of wine (score between 0 and 10), based on eleven input variables. Using the SHAP package for `Python` a local explanation of a single prediction can be made as shown in Figure 3.24. This waterfall plot indicates the marginal contribution of each input variable to reach the final model output $f(x)$, starting from the expected output $E[f(x)]$. A global explanation of the trained model can be made by displaying the importance of each feature over all data, this is depicted in Figure 3.25. In this figure, each dot indicates one instance of a feature, the horizontal position of that dot depicts the impact that feature had on the model output. The color of the dots specify whether the feature had a high or low value. From this figure observations can be made about the trained model, e.g. there is a strong correlation between high alcohol values and high model output.

This section has expressed the importance of XAI in ML. Different types of explainability techniques were highlighted alongside some examples of implementations. In Section 4.4 a description is given of how XAI was put to practice during the preliminary experiment of this thesis.

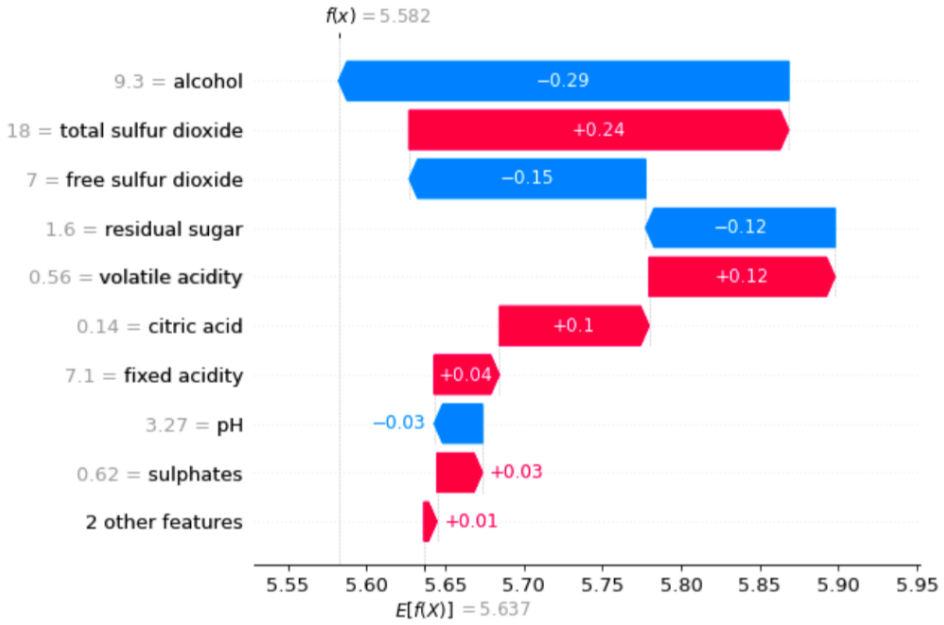


Figure 3.24: Waterfall plot generated with SHAP package for Python. This plot indicates a single instance of input variables values and how those input values contributed to the model output (i.e. local explanation). Image taken from https://medium.com/dataman-in-ai/the-shap-with-more-elegant-charts-bc3e73fa1c0c

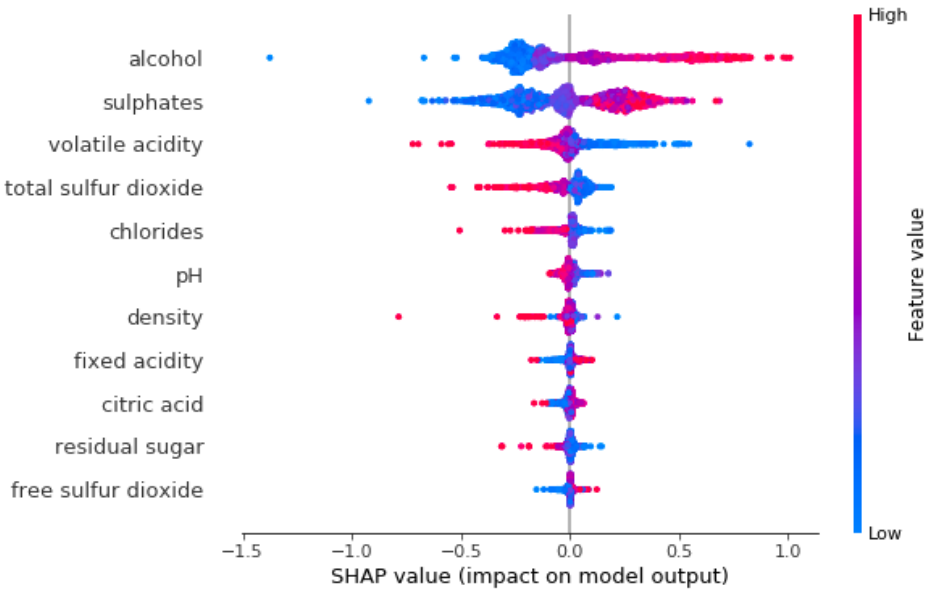


Figure 3.25: Summary plot generated with SHAP package for Python. Every dot in this plot indicates an observation of the entire training data (i.e. global explanation). The color of the dot specifies whether the feature had a high (red) or low (blue) value. The horizontal axis indicates if that feature value resulted in a higher or lower model output. Image taken from https://towardsdatascience.com/explain-your-model-with-the-shap-values-bc36aac4de3d

3.5. Chapter Takeaways

Artificial Neural Networks (ANNs) are a form of machine learning that can be used to model nonlinear and complex input-output relationships. This chapter introduced some of the key concepts of ANNs. After a brief explanation of how traditional ANNs work, some more complex popular deep network architectures were presented. Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) appear to be the most promising option for Time Series Classification (TSC).

RNNs are specifically designed to handle sequential data. An improved version of traditional RNNs, called Long Short-Term Memory (LSTM), will be used for the preliminary experiment of this thesis. This decision was made because LSTMs have a proven success in classifying pilot control behavior (Versteeg, 2019), therefore rendering them an excellent choice to get an indication of the feasibility of using ANNs to classify pilot skill level.

CNNs were designed for computer vision tasks, but have expanded to many more areas of operation, including TSC. The CNN architecture will be investigated in the main phase of this thesis, this will allow for a comparison in performance between RNNs and CNNs.

Lastly, different methods of eXplainable Artificial Intelligence (XAI) were introduced. As deep neural networks have grown in popularity, so has XAI. This is because XAI allows to get a better understanding of the internal workings of deep networks. Deep learning models are non-transparent and therefore require post-hoc explainability. Two interesting post-hoc explainability methods are: *class activation maps* that indicate discriminative regions in time traces, and *SHAP values* that quantify relative feature importance.

4 Preliminary Testing of Classifier

4.1. Preliminary Tests Objective

This preliminary experiment was not designed to find and implement an optimal deep learning model, but rather to test the viability of using a deep neural network to identify pilot skill level. Therefore the results of this preliminary experiment should be considered as a proof of concept that automatic feature extraction and classification can successfully be applied to identify pilot skill level.

The preliminary implementation of the artificial intelligence is documented in Section 4.2. Next, the preliminary data augmentation and explainable artificial intelligence results are provided in Section 4.3 and Section 4.4, respectively.

4.2. Artificial Intelligence Implementation

The following section will concern the implementation of an artificial intelligence model. Specifically, a recurrent Long Short-Term Memory (LSTM) network (as described in Section 3.2.3) will be utilized to tests its effectiveness in distinguishing pilot skill level.

The reason why LSTM were selected for this preliminary experiment is that these networks are specifically designed to handle sequential data, as has been explained in Section 3.2.3. This means that these networks are relatively easy to implement and should not require a lot of tailoring to achieve good, or at least mediocre, performance. Moreover, the work of Versteeg (2019) shows a proven effectiveness of LSTM networks to classify pilot data (of a compensatory tracking task). Although the goal of Versteeg (2019) was to classify the controlled element dynamics, and *not* the pilot skill level, it is a great starting point since the input data is very similar.

The neural network model was programmed in `Python v3.8` using the library `Keras v2.4.3` which runs on top of `Tensorflow v2.3.0`. A complete overview of used packages, versions, and dependencies of the different `Python` and `MATLAB` scripts written for this preliminary experiment is shown in Figure B.1.

The laptop used during the preliminary experiment is equipped with an NVIDIA Quadro P1000 Graphics Processing Unit (GPU). This GPU is compatible with CUDA, a parallel computing platform interface by NVIDIA. CUDA allows the use of GPU for general purpose processing. Using NVIDIA CUDA Deep Neural Network (cuDNN) library, parallel computation can be exhausted to train neural network models. CUDA v11.1.105 and cuDNN v8.0.5 were used throughout this experiment.

4.2.1. LSTM Model Settings

As previously mentioned, the artificial neural network architecture that will be used for this preliminary experiment is an LSTM model where the settings are taken from Versteeg (2019). These 'settings' in machine learning are also known as *hyperparameters*. Hyperparameters are all parameters that can be tuned to make networks train better and faster (Patterson & Gibson, 2017). In machine learning it is generally the case that these important model parameters cannot be analytically determined (Kuhn & Johnson, 2013), instead they are often found by extensive (automated) trial and error.

There are essentially two categories of hyperparameters present in this deep learning experiment: those that form the neural network model architecture (e.g. number/type of layers) and those that shape the input data (e.g. selection of input variables). The former will be discussed in this subsection, and the latter in the next.

The LSTM model structure used in the work of Versteeg (2019) was inspired by the stacked LSTM structure used in the paper by Saleh et al. (2017), combined with the rule of thumb hyperparameter settings from Reimers and Gurevych (2017). By stacking LSTM layers the network becomes deeper. This way the learned representation from the first layer can be passed on to the next layer, which creates representations at a higher level of abstraction. Each layer will take on a part of the task and pass it to the next, until finally the last layer provides the output (Hermans & Schrauwen, 2013). In precisely this fashion the LSTM model was structured as follows: an LSTM input layer; a dropout layer; another LSTM layer; another dropout layer; a dense (fully-connected) layer; and a softmax activation layer. A visual representation of this stacked network structure is shown in Figure 4.1. Each of the individual components that make up this network will be explained in the following paragraphs.

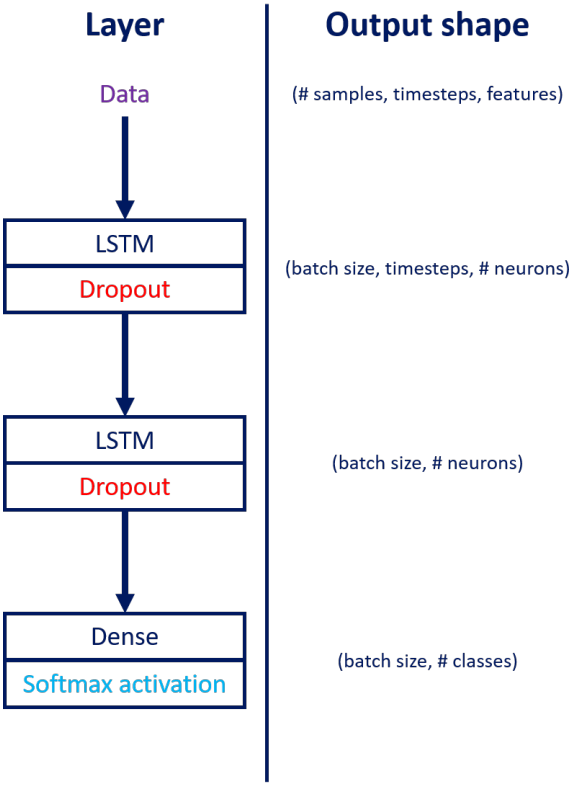


Figure 4.1: All layers and their output shape in the used stacked LSTM architecture

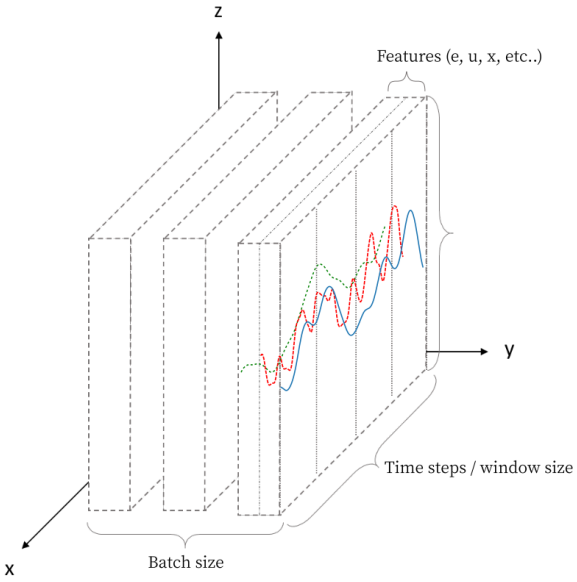


Figure 4.2: Three-dimensional LSTM input shape, image taken from (Versteeg, 2019)

Batch Size: The batch size dictates the amount of samples that are drawn from the training dataset during each *iteration* (network model parameter update). By drawing multiple samples from the training dataset at once, it takes less iterations to feed all training data to the neural network, thus speeding up the training process. One cycle of feeding all training data is called an *epoch*. For example, if there are 200 samples of training data (i.e. the training data has shape (200, timesteps, features)), then a batch size of 10 means that the network will perform 20 iterations to complete one *epoch*.

In the research of Versteeg (2019) a batch size of 100 was used to decrease training time at the (slight) cost of training stability and generalisation performance (Masters & Luschi, 2018).

Input LSTM Layer: The input LSTM layer has a three-dimensional input shape: (batch size, timesteps, features). A schematic depiction of this three-dimensional input is shown in Figure 4.2. The last two

of these three dimensions, the timesteps and features, are those that were already explained in Section 3.2.3: the sequence length and the number of input variables, respectively.

The input LSTM layer has a many-to-many mapping (as was shown in Figure 3.12), so that its output is also sequential and can thus be fed to the second LSTM layer. This means that the output shape of this layer is (batch size, timesteps, number of hidden neurons).

The amount of hidden neurons in this LSTM layer was set to 100. This value was based on (Saleh et al., 2017) and (Reimers & Gurevych, 2017).

Dropout Layer: Dropout is a regularization method introduced by Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014). This technique randomly deactivates neurons (or units) of the previous layer with a user defined probability. Deactivating (or 'dropping') these neurons, along with their connections, prevents the neurons from co-adapting too much and significantly reduces overfitting (Srivastava et al., 2014). This mechanism is only active during the training phase. A visual representation of a dropout layer is shown in Figure 4.3. The input and output shape of this layer are the same as that of the previous layer.

The dropout value in the research of Versteeg (2019) was heuristically chosen to be 0.2.

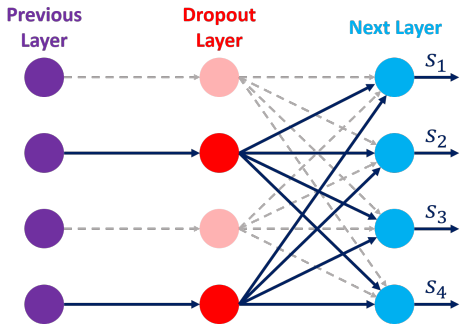


Figure 4.3: Example of dropout layer with probability of 0.5 to deactivate neurons

Second LSTM Layer: The second LSTM layer has a three-dimensional input shape that matches the output shape of the previous layer: (batch size, timesteps, number of hidden neurons). In contrast to the first LSTM layer, the second LSTM layer has a many-to-one mapping (as was shown in Figure 3.13). This was done so that the neurons in this layer only produce one output for the given input sequence, i.e. the output shape is (batch size, number of hidden neurons). The singular output of the neurons in this layer is desired because each sequence of timesteps must receive only one classification.

Like the first LSTM layer, the amount of hidden neurons was set to 100. Again, this value was based on (Saleh et al., 2017) and (Reimers & Gurevych, 2017).

Dropout Layer: The second dropout layer operates precisely the same way as the first dropout layer, i.e. random connections from the previous layer are blocked.

Again,the dropout value in the research of Versteeg (2019) was heuristically chosen to be 0.2.

Dense Layer: The dense layer fully connects the outputs of the previous layer to a specified number of outputs (just like the traditional ANN that was described in Section 3.1.1). The number of output neurons is equal to the number of classes that the network must be able to distinguish. Thus, the output shape of this layer is (batch size, number of classes).

The LSTM model in this preliminary experiment must be able to distinguish pilot behavior and categorize it into *two* classes: skilled (or 'trained'/'expert') vs. unskilled (or 'untrained'/'novice'). Therefore *two* output neurons are required in the dense layer.

Softmax Activation Layer: The last layer in the network is a softmax activation layer. As has already been described in Section 3.1.1, the softmax activation function (Eq. (3.8)) produces the probability distribution over mutually exclusive output classes. This means that for each output neuron of the dense layer, the output value s_i is transferred into an estimated probability $\hat{y}_i$ that represent the chance of the

input data belonging to that class. An example of what a dense layer in combination with a softmax activation layer looks like is shown in Figure 4.4.

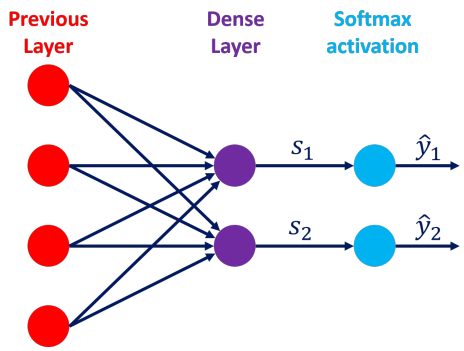


Figure 4.4: Example of dense layer and softmax activation

There are two more components present in the neural network model settings. Namely, the *loss function* and the *optimizer*. An explanation of these two components has already been provided in Section 3.1.2, but the next two paragraphs will briefly explain their implementation in this specific network.

Loss Function: As has been explained in Section 3.1.2, logistic loss functions can be used if the output of the neural network model are probabilities. The suiting logistic loss function for a network that outputs probability distribution over mutually exclusive output classes is the *categorical cross-entropy loss function* (given by Eq. (3.11)).

Optimizer: The last component to complete the neural network model is the optimizer. As has been explained in Section 3.1.2, the optimizer dictates the method that is used to update the model internal parameters (weights and biases) during training to minimize the loss. A survey by Sun, Cao, Zhu, and Zhao (2019) explains the working principle and the advantages/disadvantages of many of the popular optimizers available. It is out of the scope of this research to investigate every option, thus this paragraph will simply stick to the optimizer that was used by Versteeg (2019): *Adam*.

The Adam optimizer, introduced by Kingma and Ba (2014), has become somewhat of a standard in machine learning. This popular optimizer is an advanced *Stochastic Gradient Descent* method that combines the advantages of AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman & Hinton, 2012). It is an efficient method that only requires first-order gradients with little memory constraints. By computation of estimates of first and second moments of the gradients, individual *adaptive learning rates* are computed for different trainable parameters (Kingma & Ba, 2014).

There will be no optimization of the hyperparameters present in the Adam optimizer. This is because the default hyperparameters, as empirically found by Kingma and Ba (2014), generally show the best performance. Therefore, no mathematical description of this optimization method will be provided (this can be found in (Kingma & Ba, 2014)). However a more intuitive description, as provided by Heusel, Ramsauer, Unterthiner, Nessler, and Hochreiter (2018), is the following: "*Adam can be described as Heavy Ball with Friction (HBF), since it averages over past gradients. This averaging corresponds to a velocity that makes the generator resistant to getting pushed into small regions. Adam as an HBF method typically overshoots small local minima that correspond to mode collapse and can find flat minima which generalize well*". This description is schematically depicted in Figure 4.5.

This concludes the settings that form the neural network model architecture as was used by Versteeg (2019). All of the model settings that have been discussed are summarized in Table 4.1. In the following subsection a description will be given of what settings are applied to the input data (i.e. training/testing data).

4.2.2. Input Data Settings

This section will elaborate on the configurations that Versteeg (2019) applied to the input data. In theory the pilot data could just be fed to the neural network model without any alterations, but in practice it is favorable to adjust certain settings to obtain better performance. In the work by Versteeg (2019)

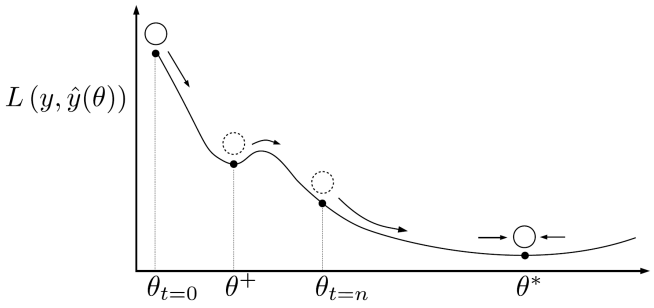


Figure 4.5: Adam optimizer can be seen as heavy ball with friction, due to its momentum it overshoots the local minimum at θ^{+} and settles at the flat minimum θ^{*}. Image taken from (Heusel, Ramsauer, Unterthiner, Nessler, & Hochreiter, 2018)

different configurations of these settings were tested on a compensatory tracking-task dataset from an experiment by Zollner, Pool, Damveld, van Paassen, and Mulder (2010). From this dataset the variables u (pilot input) and e (tracking error) were used to test the input data settings. The experiment data were split into a *training set* and a *testing set* (for reasons explained in Section 3.1.3).

The data configurations that were extensively tested and empirically optimised by Versteeg (2019) are the following: length of *window size*, magnitude of *sampling frequency*, selection of *input variables*, choice of *scaling method*, and percentage of *overlap*. These hyperparameters were empirically optimized in the order that they are listed above. The meaning of each of the hyperparameters and the results of their optimization will be presented in the following paragraphs. Additionally, some important information about *shuffling data* will be shared.

Window Size (WS): The window size dictates the length in *seconds* of each sample that is fed to, and classified by, the neural network model. This setting determines the amount of samples that can be drawn from a tracking run of the dataset. For example, if the dataset contains pilot tracking runs that are 90 seconds long, then a window size of ten seconds means that nine samples can be collected from one tracking run. There are two reasons why it is desirable to have a small window size. One, long time sequences can degrade the classification performance of some neural networks models (Sutskever, 2013). And two, a small window size (in the order of a few seconds) allows for an online application. Therefore a minimal, yet sufficient window size is desired.

The window sizes that were tested by Versteeg (2019) ranged from 0.2 seconds to 50 seconds. They were tested concurrently with the sampling frequency, as will be explained in the next paragraph.

Sampling Frequency (SF): The sampling frequency in *Hertz* determines the amount of timesteps per second of each sample that is provided to the neural network model. The combination of window size and sampling frequency shapes the input length that goes into the first LSTM layer of the neural network. As was shown in Figure 4.2, one of the dimensions of the input layer is the number of timesteps. This number of timesteps is simply found by multiplication of the window size and the sampling frequency.

One might expect that a high sampling frequency is better than a low sampling frequency (since lowering the sampling frequency means that some of the recorded data is lost). However, as was previously explained, long input sequences can degrade the performance of recurrent neural networks. Therefore, if the window size grows in magnitude, it may be desirable to lower the sampling frequency so that the length of the input sequence does not become too large (and vice versa).

As previously mentioned the sampling frequency and window size settings were concurrently tested. This test was performed by trying different combinations of window sizes and sampling frequencies on the data from Zollner et al. (2010) and recording the classification accuracy (the percentage of correctly classified samples). In these experiments both input variables u and e were fed to the neural network, where 80% of the data were used as training data and 20% of the data as validation data. The resulting validation accuracy for the different settings are shown in Figure 4.6. From these results Versteeg (2019) concluded that the best settings (smallest window size with highest accuracy) are SF = 50 Hz and WS = 1.6 s. These settings for the sampling frequency and window size were carried over to the next optimization experiment.

Table 4.1: Summary of LSTM model settings taken from (Versteeg, 2019), as have been presented in Section 4.2.1

Setting	Description	Value
Batch size	Specifies the amount of samples that are fed to the neural network model per training step.	100
Input LSTM layer (number of neurons)	Takes multivariate time sequences as input and produces equally long sequences per hidden neuron.	100
First dropout layer (dropout probabiltiy)	Probabilistically deactivates neurons of previous layer to decrease overfitting.	0.2
Second LSTM layer (number of neurons)	Takes sequences of previous LSTM layer as input and produces single scalar output per hidden neuron.	100
Second dropout layer (dropout probability)	Probabilistically deactivates neurons of previous layer to decrease overfitting.	0.2
Dense layer (number of neurons)	Fully connects outputs of previous layer to generate one output score per class.	2
Softmax activation function	Changes the output scores of the previous layer into probability distribution.	-
Categorical cross-entropy loss function	Loss function that optimizes for maximum likelihood estimation of probabilities.	-
Adam optimizer	Efficiently updates the network model trainable parameters to minimize the loss. The default hyperparameter settings of Adam optimizer (Kingma & Ba, 2014) were used in this implementation.	-

Input Variables: In principle there were three input variables available in the dataset from the compensatory tracking experiment by Zollner et al. (2010), i.e. the controlled element output x, the pilot control output u, and the tracking error e. However, only the latter two were viable options for the classification task of Versteeg (2019). This was because the goal of the neural network model was to correctly classify the pilot adaptation to different controlled element dynamics. Therefore, feeding the model the controlled element output x in combination with u would lead to a trivial solution, as the network could now discover the relationship between the two (i.e., the controlled element dynamics itself). Thus, to capture solely human control behavior, only the time traces of e and u were used (Versteeg, 2019). In the case of the current research, with the goal to classify skill behavior, addition of x would *not* lead to trivial solutions. Therefore, x is still a viable candidate as input variables.

Although only e and u could be used (Versteeg, 2019), their useful information for the LSTM model could be increased by also supplying their first-order time derivatives $\dot{e}$ and $\dot{u}$. The calculation of these time derivatives was done using second order central differences approximation in the interior points and first order accurate one-sided forward/backward differences approximations at the first and last recorded data point of each tracking run. These numerical differentiation methods are given by Eq. (C.1), Eq. (C.2), and Eq. (C.3), respectively.

With the addition of these time derivatives, there now were four input variables' time traces to choose from. Every combination of these inputs was tested simultaneously with different scaling methods by Versteeg (2019), as will be explained in the next paragraph.

Scaling Method: Applying scaling to the datasets usually helps improve convergence of back-propagation learning (LeCun, Bottou, Orr, & Müller, 2012). Additionally, it also helps to unify different sources of data, so that the trained neural network model can generalize more easily between different datasets. Three different scaling methods were tested by Versteeg (2019): *normalizing*, *standardizing*, and *robust scaling* (a visual example of each of these scaling methods applied to two different datasets is shown in Figure B.2).

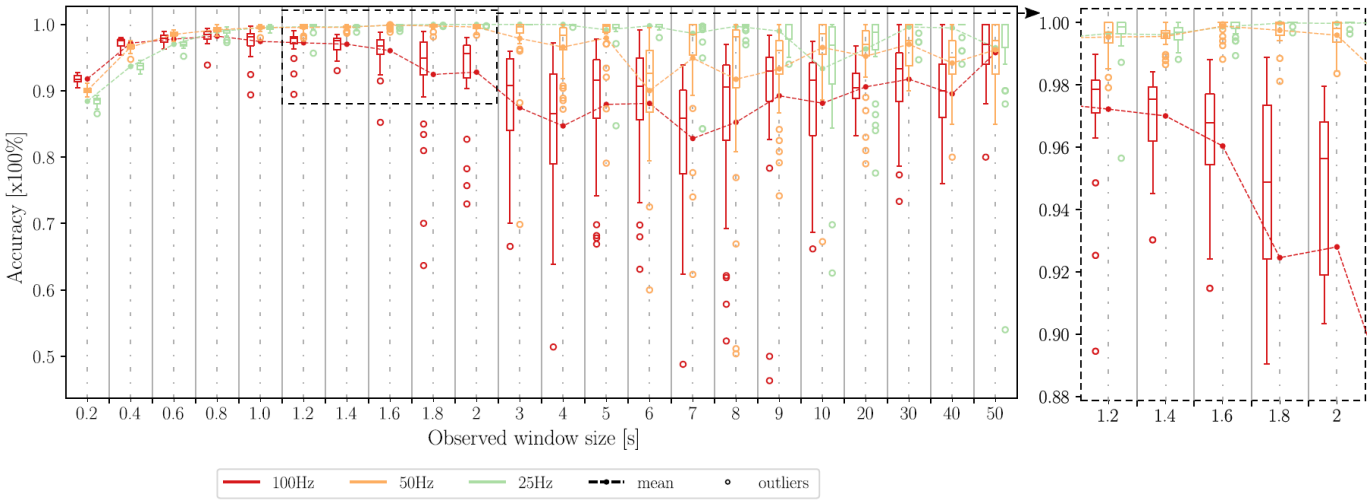


Figure 4.6: Validation accuracy for varying WS and SF. Input variables are e an u from the dataset from Zollner et al. (2010) with an 80%/20% training/validation split. Image taken from (Versteeg, 2019)

The normalization method used ensured positive and negative sign convention by scaling the entire sequence between -1 and 1 with Eq. (C.4).

Standardizing removes the mean and scales to unit variance. This is done at every timestep by subtracting the mean (of the entire sequence) and dividing the result by the standard deviation (of the entire sequence), as shown in Eq. (C.5).

Robust scaling removes the median and scales the time sequence to its inter-quantile ranges (first quantile and third quantile) (Pedregosa et al., 2011).

The scaling method can either be applied over the entire tracking run, or per sample that is fed to the neural network model. In the work of Versteeg (2019) all three scaling methods were tested over entire tracking runs, additionally normalized scaling was tested sample-wise (i.e. four different scaling options were tested).

An additional data source was needed for the validation of the different scaling methods. Considering that part of the reason to introduce scaling was to enable the neural network model to classify data from different sources. Therefore, during the simultaneous input selection and scaling method optimization, there were separate sources of data for the training phase and the testing (validation) phase. The training of the network was done on the same data from the experiment by Zollner et al. (2010). However, the testing of the network was performed on data from another tracking experiment by Lu, Pool, van Paassen, and Mulder (2015).

The results of testing different sets of input variables in combination with the explained scaling methods are shown in Figure 4.7. Again, the presented accuracy here is the *validation* accuracy (percentage of correctly classified samples) on the test data from (Lu et al., 2015). From these results it was concluded by Versteeg (2019) that the best classification accuracy is obtained for standardized scaling over the entire tracking run with the multivariate input combination of $e + \dot{e} + u$. Through brief initial testing, the current research found that using all four parameters $e + \dot{e} + u + \dot{u}$ (in combination with standardized scaling over the entire tracking run) appeared to provide better training results. Therefore, the input variables used in this preliminary research will deviate from those used by Versteeg (2019).

The extremely poor results obtained for the classification accuracy with no scaling (Figure 4.7) can be explained by a difference in units between the two data sources. Namely, the training dataset (Zollner et al., 2010) was in degrees, whereas the testing dataset (Lu et al., 2015) was in radians. This does show the importance of providing data to the network model in a consistent context. It also shows that scaling data, and thus removing the unit, helps generalization between different sources.

Overlap: The last input data setting that was optimized by Versteeg (2019) was *overlap*. This is a method that can be applied when partitioning the time traces into smaller samples using a sliding window (with the selected window size, as previously explained). Whilst the sliding window is collecting samples from the tracking run, each observed window will overlap part of the previous observed window

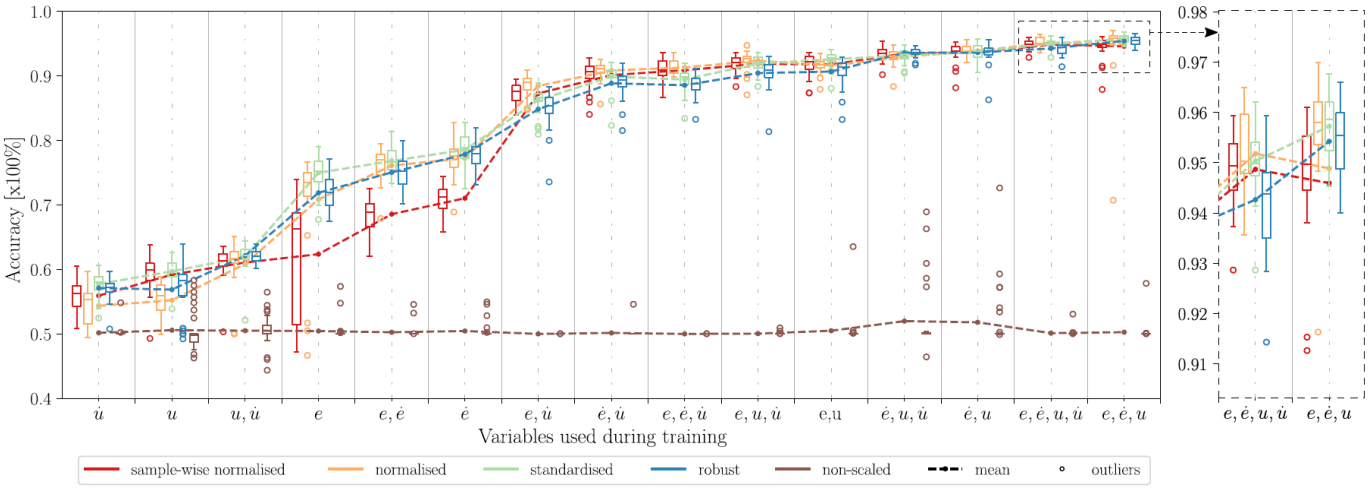


Figure 4.7: Validation accuracy for varying input variables and scaling methods. SF = 50 Hz, WS = 1.6 s. Network is trained on data from Zollner et al. (2010) and tested on data from Lu et al. (2015). Image taken from (Versteeg, 2019)

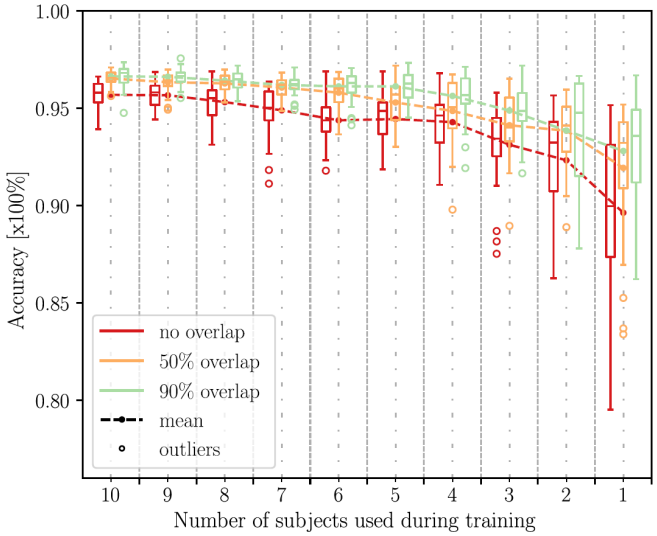


Figure 4.8: Validation accuracy with varying amounts of overlap for a reducing number of subjects. SF = 50 Hz, WS = 1.6 s. Input variables = $e + \dot{e} + u$. Network is trained on data from Zollner et al. (2010) and tested on data from Lu et al. (2015). Image taken from (Versteeg, 2019)

(Figure B.3 shows an example of what this looks like). By doing so, more samples can be drawn from the data. Therefore this can be considered a data augmentation method, as it essentially increases the amount of labeled training data available to the neural network model.

Versteeg (2019) tested the effectiveness of data augmentation using overlapping. This was done by *increasing* the amount of training samples with overlap, whilst *decreasing* the amount of available training data. The reduction of available training data was done by randomly eliminating subjects from the dataset. The results of this experiment are shown in Figure 4.8. From this figure it can be concluded that high overlap has a positive effect on classification accuracy. Especially if there is little data available, overlap appears to be able to make the biggest difference (the least amount of subjects in the dataset shows the largest accuracy difference between the different overlap settings). These results led to the decision to use 90% overlap between observed samples.

Shuffling data: Shuffling data is a crucial option that should be enabled. While training, the network learns the fastest from the most unexpected sample (LeCun et al., 2012). Ideally, each training sample would be drawn from the stack of training data randomly. Though in practice, it is usually *faster* to sequentially draw samples from the training stack. Sequentially drawing samples can, however, introduce

problems if the samples are grouped by class or come in a particular order. Presenting the training data to the neural network model in a fixed order induces overfitting. Therefore it is good practice to shuffle the stack of training data before supplying it to the neural network model (Bottou, 2012). Reshuffling the data between every epoch also prevents overfitting.

An important note must be made about the combination of shuffling data and applying overlap between samples. Namely, when these two methods are used concurrently, it is extremely important that the splitting of training/testing data is done *before* applying overlap and shuffling. To describe why this is important, consider the case where splitting the training/testing data is done *after* computing samples with overlap and shuffling these samples. In this scenario, there will be samples with overlap (thus sharing information with other samples) randomly shuffled into the stack of data. If this stack is now split into training and testing data, there will now be overlapping data between the training stack and the testing stack. This renders the testing stack useless, as the network has already seen part of this testing data in overlapping samples from the training data, i.e. unrealistically high validation performance will be found.

The steps taken to compute training/testing samples from the tracking data are shown in Figure 4.9.

This concludes the explanation of all the data related settings that were taken from the work of Versteeg (2019). A summary of all these settings, or hyperparameters, is provided in Table 4.2.

It is important to note that there are no generic optimal settings when it comes to hyperparameter tuning. The best hyperparameters will vary for every machine learning exercise. Therefore, the empirically optimized settings found by Versteeg (2019) are not necessarily optimal for the *skill level* classification task of this thesis. It is expected that the differences in control behavior between different skill levels (this thesis) are more subtle and less consistent than the differences in control behavior between different controlled element dynamics (Versteeg's thesis). However, since both research theses concern the classification of compensatory tracking run data, the optimized model settings of Versteeg (2019) are definitely a fitting starting point for this preliminary experiment.

Table 4.2: Summary of input data settings taken from (Versteeg, 2019), as have been presented in Section 4.2.2

Setting	**Description**	**Value**
Window size	Partitions tracking data into samples of specified length in seconds.	1.6 s
Sampling frequency	Determines the amount of timesteps per second of each sample. Length of each sequence (sample) is determined by multiplication of window size and sampling frequency.	50 Hz
Input variables	Set the variables that are included in the multivariate time sample provided to the LSTM model.	e + $\dot{e}$ + u + $\dot{u}$
Standardized scaling	Removes the mean and scales to unit variance.	-
Overlap	Percentage of overlap between consecutive samples.	90 %
Shuffling data	Shuffling the data before training, and between every epoch, increases learning performance.	-
Train/test split	Percentage of data that are used as training data, and percentage used as validation data.	80/20%

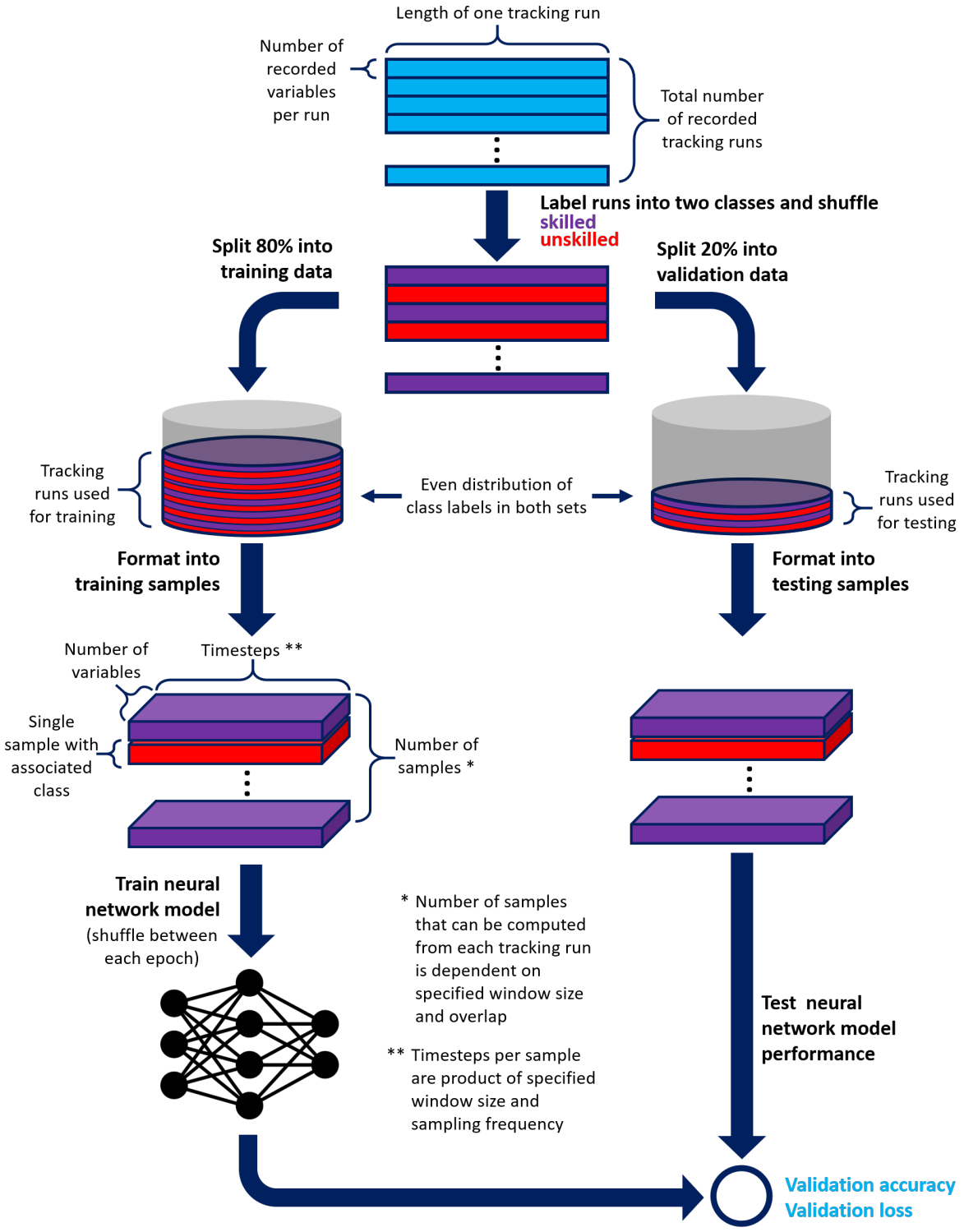


Figure 4.9: A schematic flow diagram displaying all the data handling steps taken to compute samples from the tracking data, and to train/test the neural network model. Scaling is excluded from image, this could either be done per tracking run, or per sample

4.2.3. Data Labeling

The machine learning task of this thesis can be categorized as *supervised learning*. In supervised learning a ground truth output (in this case a class) is available for the artificial neural network to learn from. By comparing the network model output to the ground truth output, a loss can be calculated. The loss can then be minimized by updating the network model parameters. This comparison with ground truth data requires *labeled* data to be provided to the network. Since all the hyperparameters are already set, this labeling of data is the last step that has to be completed before training can start.

This subsection will explain how (and why) the data can be labeled. A comparison is made between two different labeling options, and ultimately it is argued why a specific label setting is the most appropriate.

As has been discussed in Section 2.1.2 the dataset that will be used for this preliminary experiment will be taken from the research by Pool et al. (2016). It has already been argued that specifically the first 100 (training) runs of this dataset are suiting for this pilot skill level classification task. This is because this part of the tracking data contains both fully task-naive (or 'unskilled/'untrained'/'novice') participants, as well as experienced (or 'skilled'/'untrained'/'novice') participants.

Although it is known that the data contain different levels of skill, there are various ways to label what part of the data should be labeled as 'skilled' and what part should be labeled as 'unskilled'. Or, using the terminology of Section 1.1, which (multivariate) time series X_i should receive what one-hot label vector Y_i. Given that there are two classes to be distinguished, there are two options for the label Y. Namely, for a sample of *unskilled* behavior $Y = [1\ 0]^T$, and for a sample of *skilled* behavior $Y = [0\ 1]^T$. Several methods to label the data were investigated. However, only the two most promising labeling methods will be discussed in this subsection: labeling based on experience, and labeling based on performance.

On a practical note: to limit the scope of this preliminary research, only the data of the participants that did *not* receive motion feedback during the experiment by Pool et al. (2016) will be used.

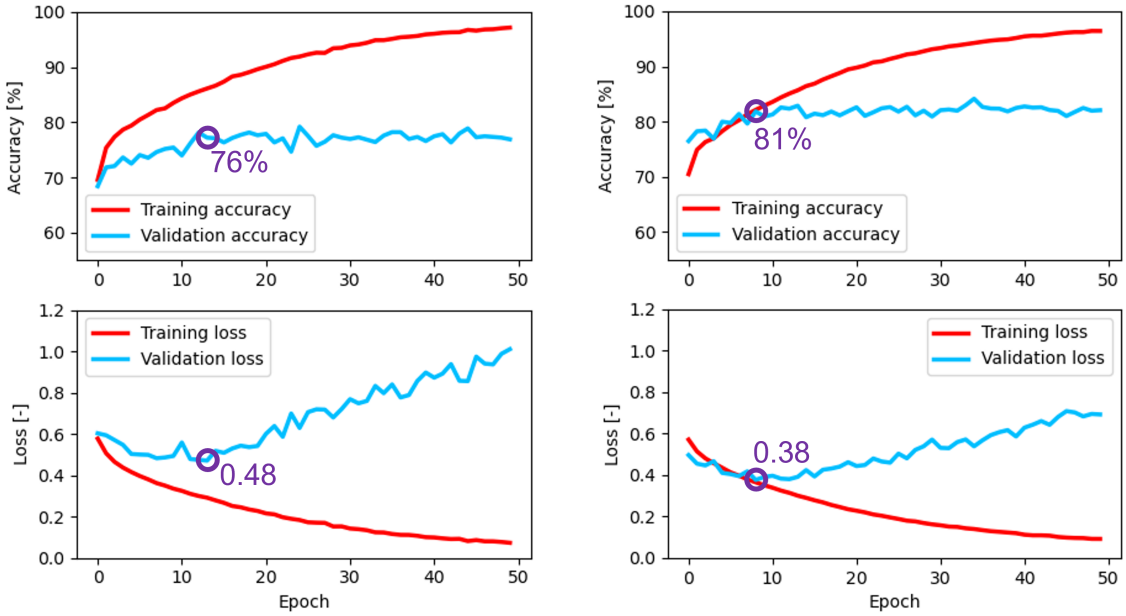
Label based on experience: With this option the class label that is added to the training data is determined by the amount of experience that the participant has. No assumption is made about the performance level of the participant, the amount of experience is simply expressed as the number of tracking runs that participant has completed. Thus, with this labeling method, the label of each time sample is solely based on the index of the tracking run that sample was taken from.

As has been mentioned in Section 3.1.3, for good training results, there should be an even amount of data samples for each class. So for the case of this research, where there are two labels (skilled/unskilled), each label should represent 50% of the data. This meant that the labeling based on experience should be done symmetrically, i.e. if the first ten tracking runs of each participant are labeled as 'unskilled', then the last ten tracking runs must be labeled as 'skilled'.

There was a trade-off when it came to selecting the appropriate number of runs belonging to each label. The purest example of an inexperienced subject is probably their first tracking run, so it would make sense to label this as 'unskilled'. Likewise, the most experienced behavior of each subject is probably found in their last tracking run, thus this could be labeled as 'skilled'. However, if only the first and last run of each subject are labeled, then there are very little data left to train the network. Now, if the first 50 runs of each subject are labeled as 'unskilled' and the last 50 runs are labeled as 'skilled', there is suddenly a whole lot more data available for training. However, now the quality of the training data is polluted, since any tracking run near the 50^{th} probably is not a good representation of either label ('unskilled'/'skilled').

This trade-off could not be solved analytically. Therefore, it was empirically tested how the number of runs belonging to each label affected the classification performance (whilst keeping all the other settings as they were listed in Table 4.1 and Table 4.2). In Figure 4.10a the learning progression per epoch can be seen if the *first five runs* of each participant are labeled as *'unskilled'* and their *last ten runs* are labeled as *'skilled'*. Figure 4.10b shows the learning progression if the first/last *ten* runs of each participant are labeled as 'unskilled'/'skilled'.

A comparison between these options can be made by inspecting the performance at minimal loss (before overfitting) and comparing the best (highest) validation accuracy and the best (lowest) validation



(a) First/last **five** runs are labeled as unskilled/skilled.

(b) First/last **ten** runs are labeled as unskilled/skilled.

Figure 4.10: Training loss, training accuracy, validation loss, and validation accuracy after each training epoch. Highest validation accuracy, and lowest validation loss are recorded as LSTM model performance measure.

loss between the differently trained networks. Note that sometimes higher validation accuracies can be seen after overfitting (i.e., with higher loss). These should be ignored, because although the network had a better performance in terms of accuracy, it was less confident about its decision (i.e., it made 'lucky' predictions).

Based on just Figure 4.10a and Figure 4.10b, one could argue that only labeling the first/last *ten* runs is the better option out of the two ($81\% > 76\%$ and $0.38 < 0.48$) . However, due to the stochastic nature of the learning process, one learning cycle is not a good indication of performance. Therefore, every setting was ran multiple times so that they could be statistically compared.

The results of running different settings multiple times are presented in Figure 4.11. Here the number on the x-axis indicates the limit of run indices that were included in each label, e.g. 10 means that the first 10 runs were labeled as unskilled, and the last 10 runs were labeled as skilled. The blue box plots indicate the best validation accuracy achieved with the respective labeling setting. The red box plots indicate the best validation loss achieved. N indicates the amount of data points in each box plot, i.e. the amount of times the network was trained with each setting.

There is quite a significant spread in performance within each labeling setting. This is partly due to the stochastic nature of parameter updating during training, but also largely due to randomly splitting the tracking runs into train/test data. This indicates a substantial sensitivity to the selection of tracking runs that are provided to the network for training. The impact of this complication is especially notable for the settings were less tracking runs are used. This makes sense, since with less runs used for training, every poorly selected run will be a bigger chunk of the training data. It is out of the scope of this preliminary experiment to further investigate why certain selections of runs work better than others, but this should be addressed in the main phase of this thesis. A potential method to examine this problem is by cross-validation (Berrar, 2019).

As was speculated, Figure 4.11 shows that the classification performance deteriorates as the limit of included run indices increases. It is difficult to select an 'optimal' setting from this preliminary experiment, but it can be seen that a more strict definition of unskilled/skilled (i.e. less runs included per label) leads to better classification performance (higher validation accuracy). More research will

be required to draw any definite conclusions about what the best labeling setting is. However, given the earlier discussed uncertain performance at lower run limits, and the poor performance at higher run limits, it appears the best option is to use somewhere between ten and twenty runs to produce the labels. Therefore, for the remained of the preliminary experiment, only the first/last **fifteen** runs of each participant will be used as labeled training data.

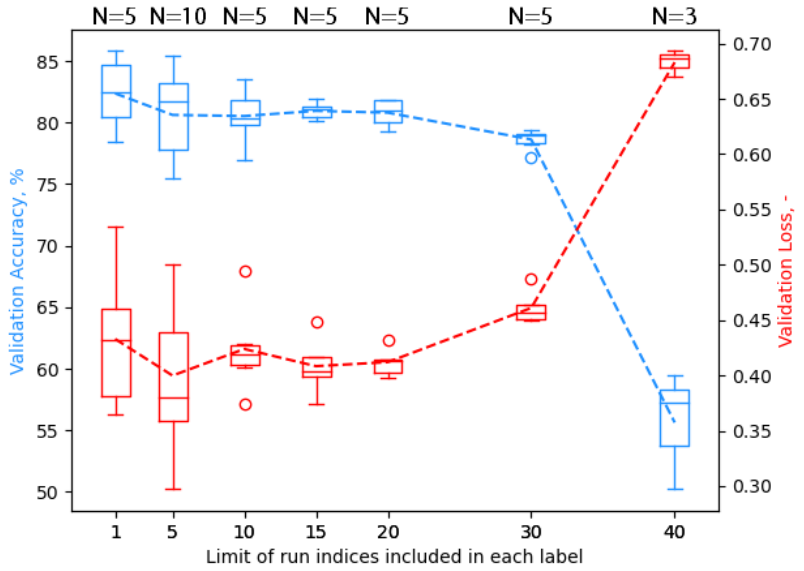


Figure 4.11: Dual axis box plot showing how the validation loss and the validation accuracy are affected by the limit of run indices included in each label. N denotes the number of data points in each box plot

Label based on performance: In contrast to the previous labeling method, this method *will* assume the subject's skill level based on their performance. As has already been observed in Section 2.1.2, there are two obvious learning patterns visible in the training data as the subjects gain more experience: the variance of the tracking error σ_e^2 *reduces*, and the variance of the subject control output σ_u^2 *increases*. These patterns can be utilized to quantify the acquisition of participants' skill, and thereon base the associated class label.

A research by Wijlens, Zaal, and Pool (2020) evaluated tracking run performance (and control activity) in terms of the Root Mean Square (RMS) of the error and the control signals, RMS(e) and RMS(u), respectively. The current research will test the use of RMS(e) of each tracking run as a measure to label said tracking run as 'unskilled' or 'skilled'. It was also tested to base labels on RMS(u), but this resulted in poor training results.

The RMS(e) of every tracking run in the dataset is shown (blue dots) in Figure 4.12. The average RMS(e) at every run index is also shown (red dots), to visualize acquisition of skill by the participants. By comparing every individual tracking run's RMS(e) to the median value, a label can be generated for each run. This way any tracking run that is below the median, will be labeled as 'skilled', and vice-versa. It was found that using this performance based labeling method resulted in a higher classification accuracy of the trained network. A comparison of performance of the two different labeling methods is shown in Figure 4.13. From this graph it can be seen that the performance based labeling method increases the classification accuracy of the network by about 8%.

Though these results may seem promising on the surface, there is a catch. The labeling method dictates *what* the network is able to recognize, i.e. using RMS(e) to label the training data means that the network will become good at predicting whether a sample belongs to a tracking run with a low or high RMS(e). This does *not* necessarily mean that the network has become good at predicting whether a sample belongs to a tracking run of a trained or untrained pilot.

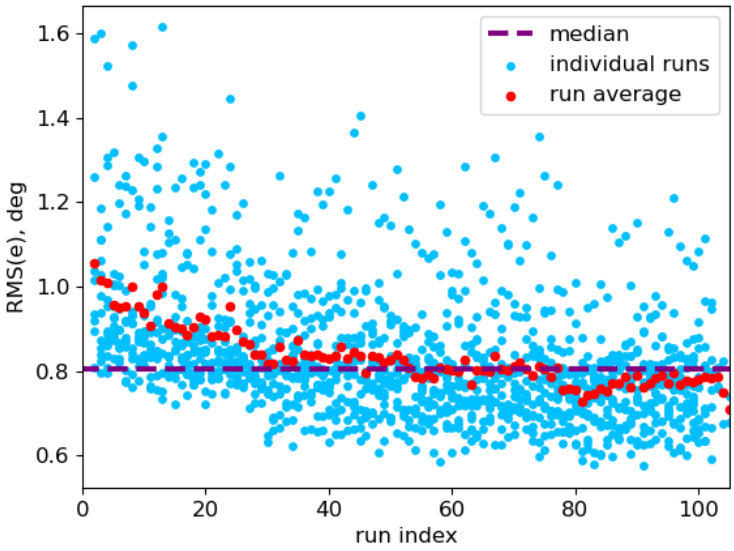


Figure 4.12: RMS(e) of all tracking runs indicated by blue dots. The average RMS(e) of all participants at a certain run index is shown in red. The median RMS(e) value is indicated by the purple dotted line.

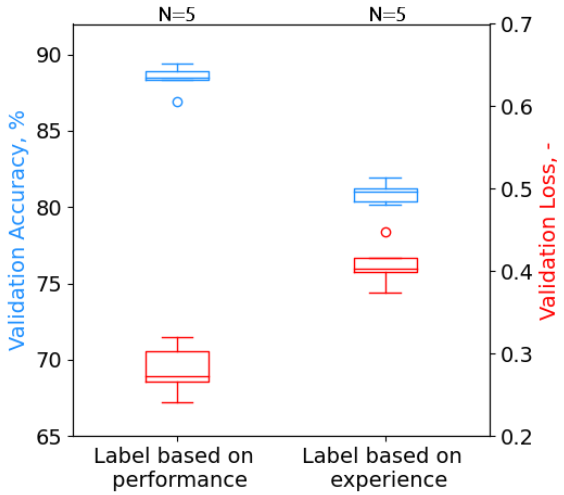


Figure 4.13: A comparison of validation accuracy and validation loss, between the two different labeling methods that have been discussed.

To visualize the impact the labeling method has on the model output, consider Figure 4.14 and Figure 4.15. In these two figures every dot indicates a separate tracking run, the color of the dot specifies the average classification of the numerous samples in that run (remember that the network classifies samples of a specified window size, not entire tracking runs). The vertical position of the dots shows the variance of the tracking error σ_e^2, whereas the horizontal position depicts the run index. Lastly, the bar chart in the lower half of these figures shows the percentage of samples that were classified as *skilled*, in bins of five tracking runs.

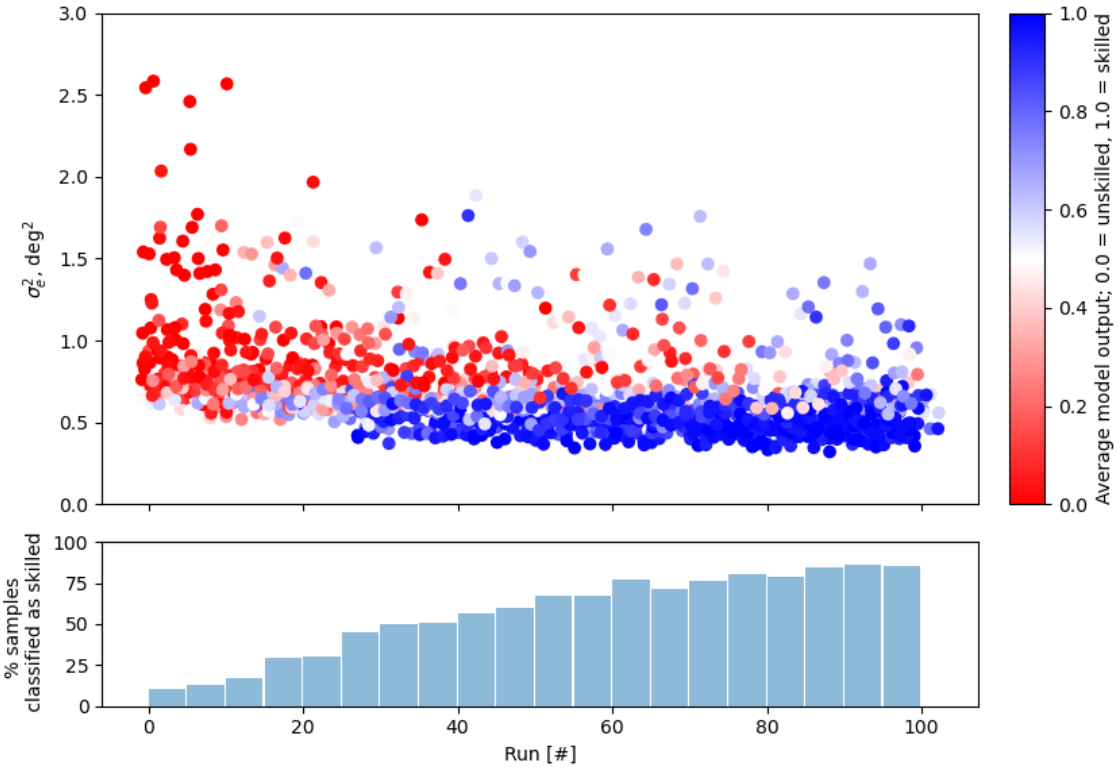


Figure 4.14: Average model output for every tracking run. This model has been trained with the *experienced based labels* (i.e. label based on run index).

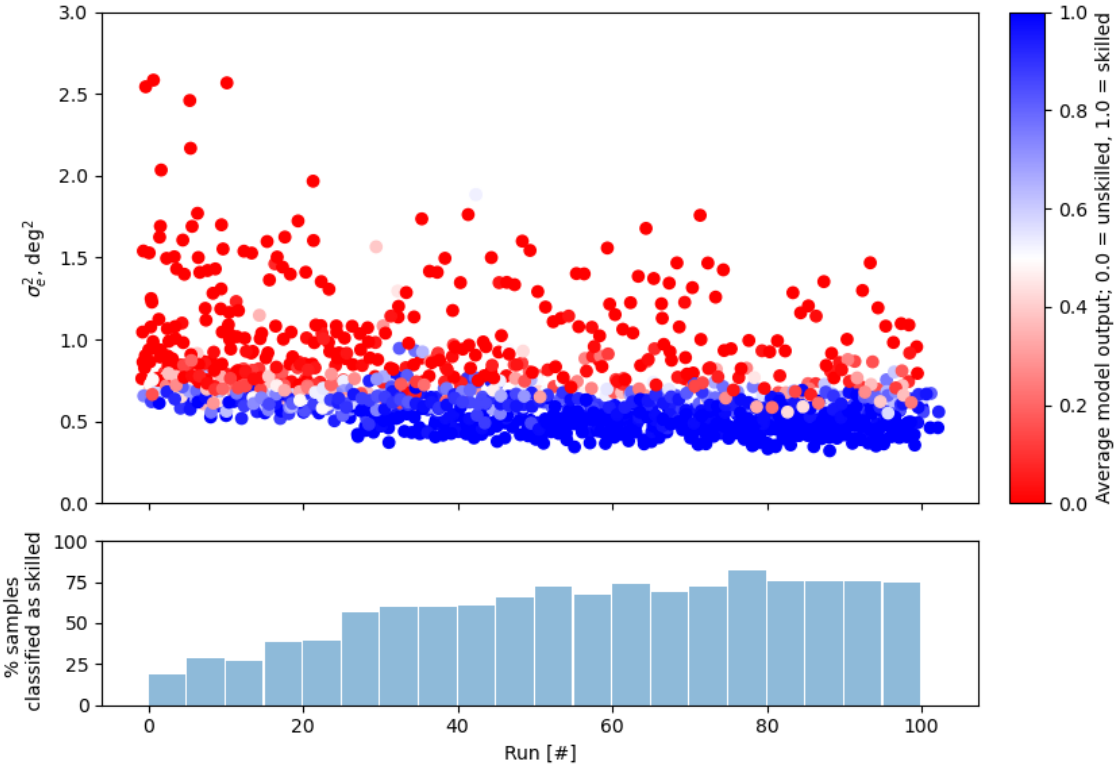


Figure 4.15: Average model output for every tracking run. This model has been trained with the *performance based labels* (i.e. label based on RMS(e)).

The output of the network that was trained using labels based on experience is shown in Figure 4.14, the result of using labels based on performance is shown in Figure 4.15. Inspection of these figures clarifies the notion that the labeling method dictates *what* the network is able to recognize. This can be observed by the horizontal distribution of colors in Figure 4.14 (left is predominantly red, right is predominantly blue), and the vertical distribution in Figure 4.15 (top is predominantly red, bottom is predominantly blue). Namely, the horizontal distribution indicates that the network is good at distinguishing the amount of experience a pilot has, whereas the vertical distribution means the network is good at distinguishing low and high tracking error. When comparing the bar charts, it can also very clearly be seen that Figure 4.14 shows a strong correlation between run index and percentage of samples classified as skilled. It is especially impressive that this correlation is also visible between run index 15 and 85, since this network was only trained with the first and last fifteen tracking runs of each participant. Inspection of the bar chart in Figure 4.15 shows that this correlation is far less present, as the second half of the bars are almost equal in height.

There is no right or wrong option between choosing either one of the labeling options, it is a matter of preference. In light of the goal of this thesis, however, the option of labeling based on *experience* seems most suitable. This is because this method has an unbiased definition of skill level, i.e. there is no arguing that a pilot performing a tracking task for the first time is inexperienced. Contrarily, the labeling based on *performance* option makes the assumption that pilot skill level is only determined by tracking error. This is a disputable approach for two reasons. 1) It ignores all the other variables that were recorded during the tracking runs and solely bases the class on the error signal. 2) There can be cases where a 'good' pilot has a high tracking error, even though they are carrying out *skilled* safe control behavior (and vice-versa). These cases can be identified in Figure 4.15 as the completely red dots in the last tracking runs, or the blue dots at the very first tracking runs. In comparison, Figure 4.14 shows far fewer of these cases.

4.2.4. Sensitivity Analysis

As a last step of the LSTM model implementation, a brief sensitivity analysis was conducted. The LSTM model settings (Table 4.1) were left intact, whilst the influence of the data settings (Table 4.2) was tested. Only the scalable data settings were tested, i.e. the *window size*, *sampling frequency*, and *overlap*.

The sensitivity analysis was conducted by recording the new validation accuracy of the model if *one* of the three investigated settings was changed, while leaving the other settings unaltered. To account for different training results due to the stochastic nature of the training algorithm, every combination of settings was tested five times under five fixed random seeds. Fixed random seeds were used to ensure that every tested setting was dealt the same five random selections of training/testing samples.

The results of the described sensitivity analysis are shown in Figure 4.16. The LSTM model appears to be relatively insensitive to doubling (3.2 s) or halving (0.8 s) the *window size*, more extreme values will have to be tested to deduce a correlation. Interestingly, using a higher *sampling frequency* actually appears to increase classification performance, contradicting the findings by Versteeg (2019) (as were shown in Figure 4.6). This underlines the unpredictability of training artificial networks and highlights that 'one-size-fits-all' is often not the case for these algorithms. Lastly, a very strong correlation between *overlap* and validation accuracy can be observed. A high overlap, which is essentially a data augmentation method, significantly increases classification performance. This increased performance can be explained by the larger amount of training samples available to the network when using a high overlap between consecutively collected samples with the sliding window technique that has been explained in Section 4.2.2.

These results only serve as an *indication* of model sensitivity. In the main phase of this thesis, a more thorough analysis should be conducted. For example, each setting should be tested more often to further reduce the noise in the results and get statistically more meaningful results. Furthermore, a wider range of values for each setting should be investigated. Lastly, the effect of changing multiple settings concurrently should be studied.

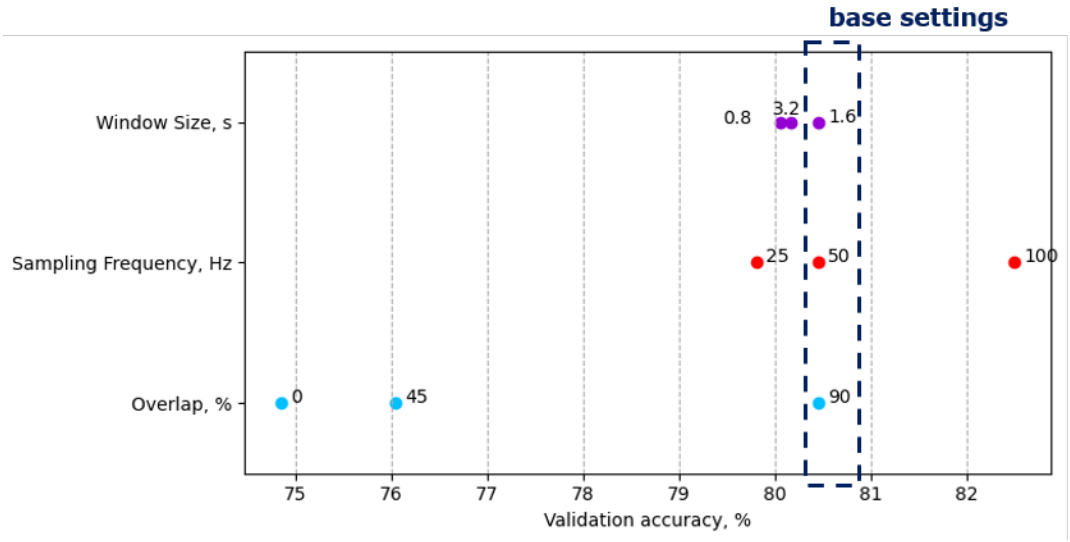


Figure 4.16: Average validation accuracy after training the LSTM model with different data settings.

4.2.5. Lowering Data Resolution

During the tests performed to gather preliminary results, the neural network training software repeatedly crashed. Most of the time these crashes appeared be the consequence of computer memory issues. Specifically, the VRAM became overfull, causing the GPU to delay, which in turn disconnected the GPU from the Kernel that was running the training algorithm. Additionally, loading in multiple data sets occasionally resulted in a full workspace due to shortage of RAM, also causing memory errors. These problems could potentially be overcome by lowering the amount of bits per stored data point.

The default size of the data points is 8 bytes (64 bits) per stored decimal number. The amount of bits per number dictates the resolution of the stored number. By lowering the resolution, part of the information of each data point is lost. This could affect the training results.

An experiment was conducted to empirically determine the influence of lowering the amount of bits per data point. Three `Python` data types were tested: 64 bit float, 32 bit float, and 16 bit float. Again, every different (data type) setting was tested five times with five fixed random seeds. All other settings were kept the same as they were presented in Section 4.2.1 and Section 4.2.2. The results of this experiment are shown in Table 4.3. Note that the validation accuracy and validation loss seem unaffected by the reduced amount of bits per data point. The small performance differences that are found are likely due to the random parameter updating algorithm during training.

For the remainder of the report 32 bit floats were used in the experiments, this greatly reduced software crashes and also slightly decreased computation time (due to faster reading/writing of the smaller data type). Although the 16 bits was also a viable option, the 32 bits option was selected. This is because 32 bits were sufficiently low to overcome the problems, and therefore further lowering the bits per number would only increase the chances of running into any unforeseen unwanted side effects of resolution reduction.

Table 4.3: Influence of using less bits per stored data point (lower resolution of decimal number).
* Reported total data size is under the data settings presented in Table 4.2.

Data type used	Total train and test data size*	Validation accuracy	Validation loss
64 bit float	814 Megabytes	80.36%	0.430
32 bit float	407 Megabytes	80.50%	0.427
16 bit float	203 Megabytes	80.43%	0.430

4.2.6. Takeaways

This section discussed the implementation of a stacked LSTM model to classify pilot skill level. The model architecture, along with hyperparameter values, are summarized in Table 4.1 and Table 4.2. The performance, in terms of validation accuracy, is highly dependent on the definition of the class labels (i.e. what data are labeled as 'skilled' or 'unskilled').

When basing the label on Root Mean Square of the participant's tracking error, a validation accuracy around 88% can be achieved. However, this method only considers the tracking error to produce the class label, ignoring the other input variables. Subsequently, this results in some *trained* participants being classified as 'unskilled' and vice-versa.

Labeling the tracking samples based on experience (i.e. the run index) results in a validation accuracy slightly higher than 80%. Although this is a lower accuracy, it seems to be better at capturing the learning curve of the participants.

Lastly, a brief sensitivity analysis was conducted. From this analysis it was found that the performance is specifically sensitive to the *overlap* setting. The window size seemed to have little influence. Increasing the sampling frequency to 100 Hz resulted in about 2% higher validation accuracy.

Storing the data in a smaller memory bit format appeared to have no influence on the training performance. However, this significantly reduces required computer memory and decreases the chance of the code crashing.

4.3. Cybernetic Data Augmentation Implementation

This section will discuss the preliminary results of using a pilot model to generate additional training data. First the MATLAB implementation of a quasi-linear pilot model is described. Next, a preliminary experiment is conducted to test the feasibility of utilizing this model as a data augmentation method.

4.3.1. Cybernetic Pilot Model

The theoretical background of the pilot model used in this preliminary experiment has already been provided in Section 2.2.2. Therefore, this section will concern a pragmatical description of putting the provided theory to practice. The implementation of the pilot model in MATLAB will be discussed in two parts: 1) the steps taken to create the pilot model and 2) the steps taken to create the remnant signal.

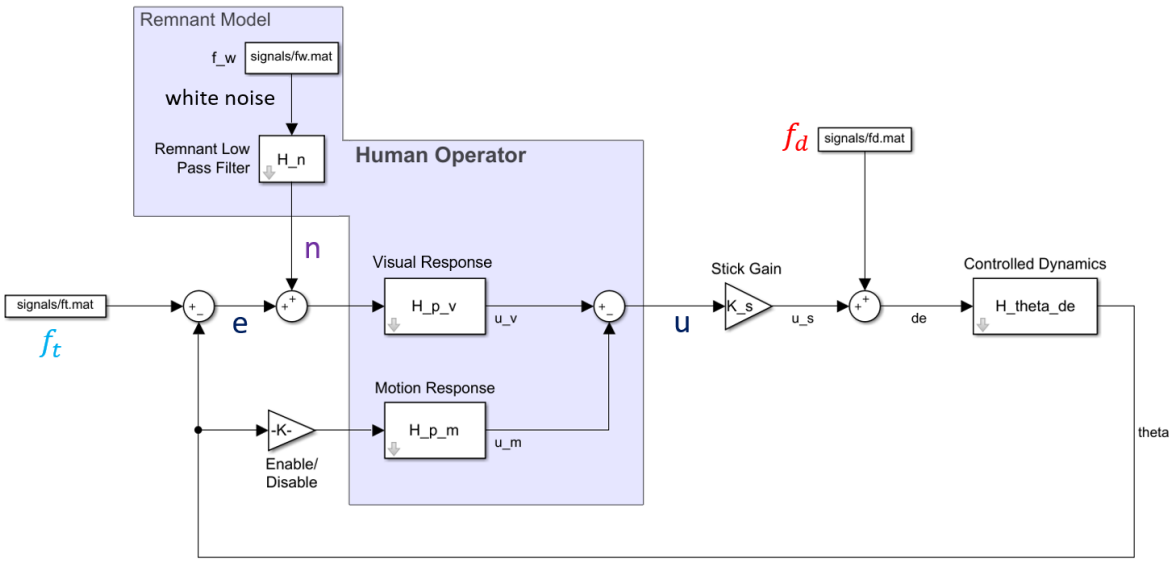


Figure 4.17: Screenshot of the quasi-linear pilot model as it has been implemented in Simulink.

Pilot model: The implementation of the pilot model described in Section 2.2.2 was done in MATLAB using Simulink. A screenshot of the final Simulink model is shown in Figure 4.17. This model

closely resembles the schematic depiction that was provided in Figure 2.6, the only difference is the position in the loop where the remnant is injected (this will be further discussed in the next paragraph).

The transfer functions H_{θ,δ_e}, H_{p_v}, and H_{p_m} are given by Eq. (2.1), Eq. (2.2), and Eq. (2.4), respectively. As will be discussed in Section 4.3.2, the model parameters in these transfer functions matched the estimated parameters of individual tracking runs as found by Pool et al. (2016). Throughout the preliminary experiment the motion response was disabled, as only the training runs without motion feedback were simulated.

The forcing functions f_t and f_d were identical to those used in the experiment by Pool et al. (2016). This means that f_t and f_d were generated as sum-of-sine signals using Eq. (4.1). Here the amplitudes $A_{d,t}$, frequencies $\omega_{d,t}$, and phases $\phi_{d,t}$ of the individual sinosoids are given by Table 4.4. The sinusoids' frequencies $\omega_{d,t}$ are integer multiples of the measurement time base frequency $\omega_m = 2\pi/81.92 = 0.0767$ rad/s (Pool et al., 2016).

$$f_{d,t}(t) = \sum_{k=1}^{N_{d,t}} A_{d,t}[k] \sin\left(\omega_{d,t}[k]t + \phi_{d,t}[k]\right) \quad \text{with} \quad \omega_{d,t}[k] = n_{d,t}[k]\omega_m \tag{4.1}$$

Table 4.4: Amplitudes, frequencies, and phases used to generate forcing functions. Data taken from (Pool, Harder, & van Paassen, 2016).

Disturbance signal f_d								Target signal f_t							
n_d	ω_d, rad/s	A_d, deg	$\phi_{d,1}$, rad	$\phi_{d,2}$, rad	$\phi_{d,3}$, rad	$\phi_{d,4}$, rad	$\phi_{d,5}$, rad	n_t	ω_t, rad	A_t, deg	$\phi_{t,1}$, rad	$\phi_{t,2}$, rad	$\phi_{t,3}$, rad	$\phi_{t,4}$, rad	$\phi_{t,5}$, rad
2	0.15	0.10	2.74	−0.02	−0.90	−0.71	−0.58	5	0.38	0.51	5.68	3.99	3.92	6.00	4.39
3	0.23	0.15	3.92	1.79	−1.55	2.02	3.67	6	0.46	0.49	0.83	4.35	1.42	5.23	5.77
9	0.69	0.30	3.97	−1.60	1.87	1.41	0.80	13	1.00	0.34	0.54	5.35	5.17	4.75	4.93
10	0.77	0.30	1.65	2.60	−0.02	1.90	4.00	14	1.07	0.33	1.14	5.92	1.57	6.28	4.23
22	1.69	0.19	3.60	2.30	0.30	−0.55	3.45	27	2.07	0.16	2.93	3.84	4.05	2.88	4.01
23	1.76	0.18	−1.33	−1.74	2.04	0.02	−0.58	28	2.15	0.15	2.83	3.48	6.24	1.23	2.39
36	2.76	0.12	4.76	4.09	4.02	0.29	−0.82	41	3.14	0.08	6.02	4.99	4.04	1.21	2.91
37	2.84	0.12	0.18	1.51	0.27	0.88	−0.65	42	3.22	0.08	1.74	4.97	2.71	0.29	0.46
49	3.76	0.12	2.55	1.90	3.28	0.08	0.58	53	4.07	0.06	3.90	4.26	1.71	1.88	2.56
50	3.83	0.12	0.23	2.89	4.65	4.93	1.96	54	4.14	0.05	0.74	1.00	4.03	4.62	2.08
69	5.29	0.15	0.93	3.47	0.63	0.14	4.37	73	5.60	0.03	5.65	4.69	0.67	0.89	4.56
70	5.37	0.15	3.49	3.97	0.97	0.75	4.63	74	5.68	0.03	3.70	5.01	5.85	0.97	3.33
97	7.44	0.20	2.84	3.30	5.77	3.29	2.91	103	7.90	0.02	3.63	5.44	5.13	0.74	1.43
99	7.59	0.20	5.12	5.88	5.65	2.52	3.80	104	7.98	0.02	1.42	5.78	4.14	5.81	1.44
135	10.35	0.28	0.54	5.29	1.65	1.28	−0.02	139	10.66	0.02	3.64	0.86	2.66	5.21	5.97
136	10.43	0.28	5.60	5.59	1.04	5.22	4.45	140	10.74	0.02	5.94	2.05	5.65	1.08	4.55
169	12.96	0.37	5.89	2.83	1.22	2.93	1.34	193	14.80	0.01	4.74	2.03	4.28	0.70	5.76
170	13.04	0.37	0.88	6.03	0.98	0.70	3.49	194	14.88	0.01	3.72	2.88	1.92	2.81	5.28
224	17.18	0.55	1.77	3.84	1.48	5.28	0.36	229	17.56	0.01	4.32	3.01	1.03	4.44	2.67
225	17.26	0.56	2.46	0.69	1.49	0.95	1.99	230	17.64	0.01	2.15	2.93	3.21	0.53	2.50

Remnant: As has been discussed in Section 2.2.2, the remnant n can be modeled as filtered white noise (colored noise) injected at the error. As proposed by Levison et al. (1969), a first-order low-pass filter (Eq. (2.6)) can be used to generate the colored noise. Three steps were taken to obtain the filter parameters K and T_l: 1) estimate the control output power-spectral density due to remnant $S_{uu_n}(j\omega)$, 2) calculate the power-spectral density of the remnant injected at the error $S_{nn_e}(j\omega)$, and 3) use a least-squares cost function to estimate K and T_l (van der El et al., 2019). The following paragraphs will demonstrate these three steps.

Step 1) Estimate S_{uu_n}: This step is based on the premise that the control output power-spectral density function S_{uu} is the sum of the output spectra due to target, disturbance, and remnant (i.e. Eq. (4.2)). This is under the assumption that these signals are linearly independent (Levison et al., 1969). The spectrum $S_{uu}(j\omega)$ can be estimated from discrete time measurements using Eq. (4.3) (van der El et al., 2019). Here L is the number of recorded time steps, f_s the sampling frequency in Hz, and $U(j\omega)$ is the Discrete Fourier Transform of the control output $u(t)$. To decrease noise in the power-spectral density function, S_{uu} was calculated as the average S_{uu} of five different runs.

Once S_{uu} is computed, S_{uu_d} and S_{uu_t} can be identified as the power-spectral density at the target and disturbance frequencies, respectively. S_{uu_n} can be recognized as the power-spectral density

outside of the forcing function frequencies. Interpolation was used to also estimate S_{uu_n} at $\omega_{d,t}$. An example of the above is depicted in Figure 4.18.

$$S_{uu}(j\omega) = S_{uu_t}(j\omega) + S_{uu_d}(j\omega) + S_{uu_n}(j\omega) \tag{4.2}$$

$$S_{uu}(j\omega) = \frac{1}{f_s L}\left|U(j\omega)\right|^2 \tag{4.3}$$

Step 2) Calculate S_{nn_e}: The power-spectral density of the remnant injected at the error can be estimated at the target and disturbance frequencies, using Eq. (2.8). The S_{nn_e} estimates at the disturbance frequencies appeared inaccurate and resulted in unstable results at step 3. Therefore only the $S_{nn_e}(j\omega_t)$ will be used in the next step.

Step 3) Approximate low-pass filter parameters: In the third and final step, the model parameters of the low-pass filter are determined. This is done by first rewriting Eq. (2.7) to Eq. (4.4), and then applying a least squares cost function to fit Eq. (4.4) to the estimated remnant S_{nn_e}. An example of this approach is shown in Figure 4.19.

$$\begin{aligned} |H_n| &= K\left|\frac{1}{1+T_l j\omega}\right| = K\frac{1}{\sqrt{1+T_l^2\omega^2}} \\ \rightarrow |H_n|^2 &= K^2\frac{1}{1+T_l^2\omega^2} \end{aligned} \tag{4.4}$$

Two remnant settings were calculated for each participant: an *inexperienced* setting to simulate unskilled behavior, and an *experienced* setting to simulate skilled behavior. The inexperienced settings are based on the first five runs of each participants, and the experienced settings on the last five runs. The found remnant settings are summarized in Figure 4.20.

From Figure 4.20a it can be seen that the low-pass filter gain K decreases as the participants gain more experience, this means that the amplitude of the remnant signals after training. Similarly, the low-pass filter lag time-constant T_l declines too, as is shown in Figure 4.20b. This increases the break frequency ($\omega_b = 1/T_l$) of the remnant signal, meaning that trained participants perform nonlinear behavior at a higher frequency than untrained participants.

4.3.2. Data Augmentation

This section will address the utilization of the designed pilot model as a tool for data augmentation. Throughout this section, the *experience* based labeling will be used with the first and last fifteen tracking runs of each (simulated) participant.

The goal of this part of the preliminary experiment is not to determine the added value of the proposed data augmentation method (e.g. in terms of performance increase). Therefore, this preliminary stage will not yet conduct an experiment to address the usefulness of this data augmentation in certain scenarios. Instead, only a feasibility check of the proposed method is conducted. This feasibility check is done by investigating whether an artificial neural network, that is trained solely on data generated with the pilot model, can correctly classify time traces of human pilots. In short: training data = generated by pilot model, and validation data = actual participants' tracking runs.

To execute above mentioned feasibility test, pilot data had to be simulated with the built pilot model. For every recorded tracking run of the real participants, a modeled counterpart was generated. This was done by employing the pilot model with derived model parameters of each participant for every tracking run (as found by Pool et al. (2016)), and providing the resulting pilot model with the same forcing functions that the real participants received. It was also tested to generate pilot data with 'generic model' parameters (i.e. one set of model parameters that represented unskilled behavior and one set model that represented skilled behavior). However, it was found that using the actual model parameter estimates of each participant for every individual tracking run, to train the neural network, resulted in better validation accuracy.

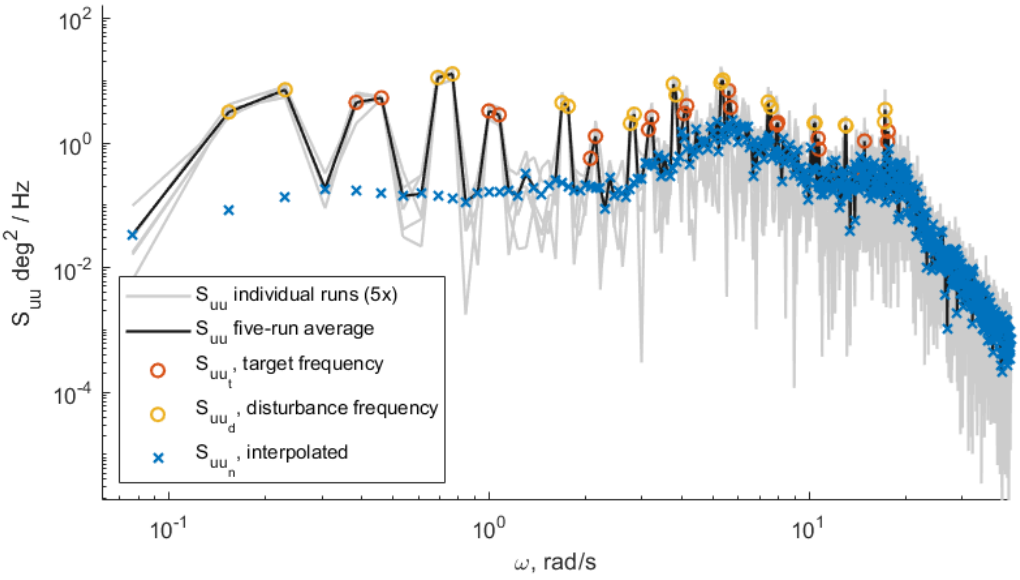


Figure 4.18: Control output power-spectral density of a single participant in the described compensatory tracking experiment.

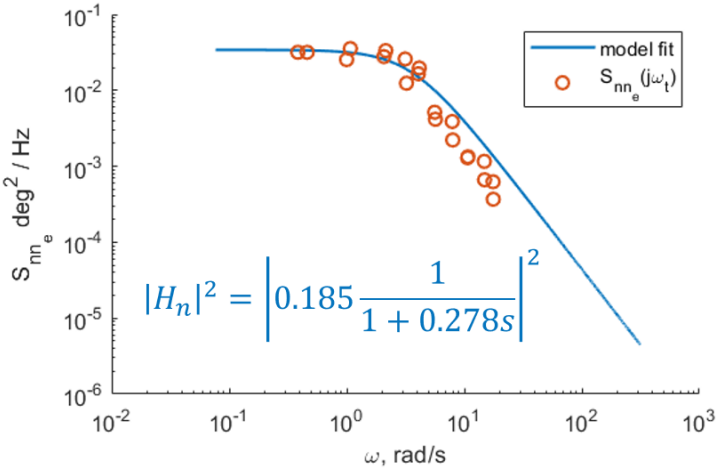
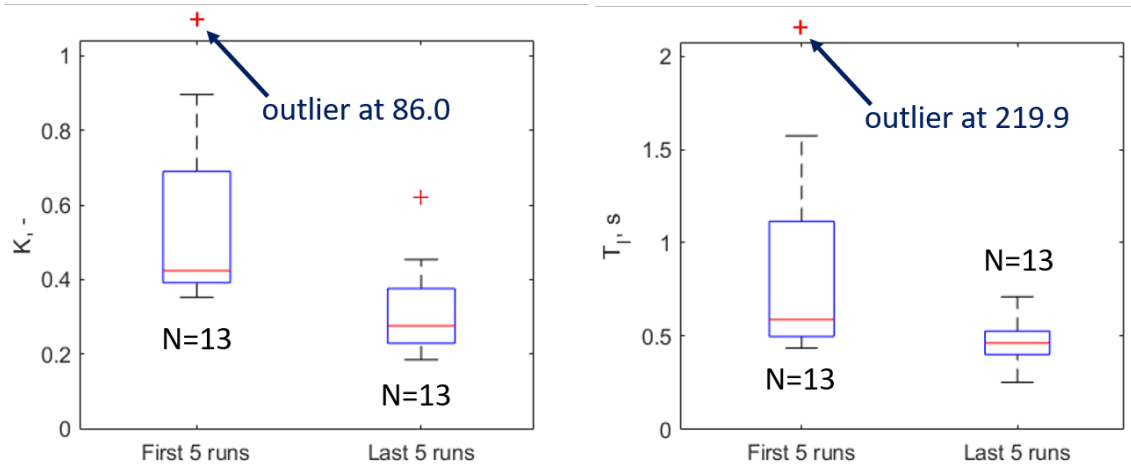


Figure 4.19: Remnant model fit for a single participant in the described compensatory tracking experiment.

As previously mentioned, for each participant only two remnant settings were computed (inexperienced setting and experienced setting). This meant that the inexperienced setting was used to simulate the first 50 runs, and the experienced setting to simulate the last 50 runs.

A comparison, in terms of variance, between the real pilot behavior and the simulated behavior is shown in Figure 4.21. This figure confirms that, on average, the simulated signals are similar to the real recorded signals. At run index 50, a gap can be observed between the average of simulated runs. This gap is likely caused by the fact that there are only two remnant settings for each participant. For the main phase of this thesis, it may be beneficial to have a more gradually changing remnant model. However, it is not expected that this will have a major impact on the network training results. This is because only the first and last fifteen runs are used to train the network (as has been explained in Section 4.2.3), and it appears that the simulated behavior is relatively accurate for those runs.

An example of time traces of simulated behavior versus real pilot behavior is displayed in Figure 4.22. The influence of including/excluding remnant in the pilot model is also depicted. This example shows that, although there is a noticeable similarity between simulation and reality, the simulated



(a) Empirically determined K values of remnant model.

(b) Empirically determined T_l values of remnant model.

Figure 4.20: Remnant model filter parameters for all 13 participants. Each participant has filter parameters for their first five runs and their last five runs.

behavior does not perfectly replicate the real behavior. This is likely because, unlike humans, the pilot model performs (quasi) linear time invariant behavior. Although the remnant adds some nonlinear control output, it does not predict the time varying behavior of the human participant. However, the difference between the real and the simulated signal is not necessarily a problem, as the simulated behavior may still be representative of unskilled/skilled behavior.

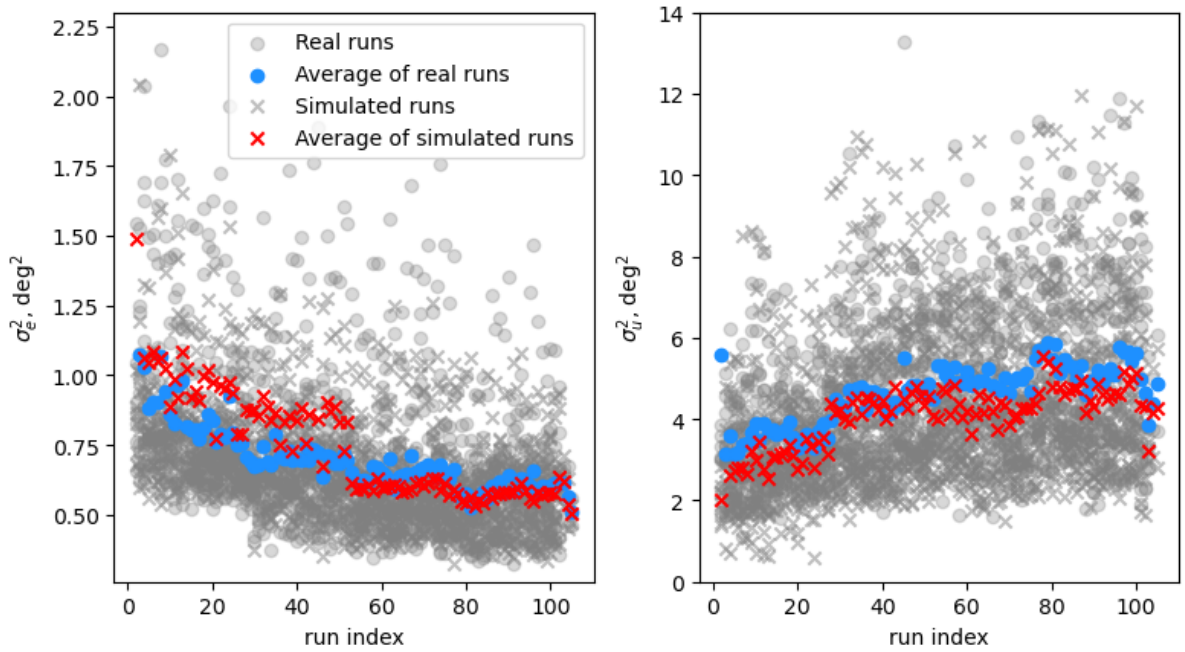


Figure 4.21: Tracking error variance σ_e^2 and control output variance σ_u^2, comparison between real runs and simulated runs.

As previously mentioned, to test the feasibility of training the neural network on simulated data, an experiment was conducted. In this experiment the training data was simulated using the pilot model, whilst the validation data was set to be actual pilot tracking behavior. All other training settings were kept as they have been presented in Table 4.1 and Table 4.2. Simultaneously, the influence of the remnant was tested by scaling the remnant signal with a gain ranging from zero to two. Fixed random seeds

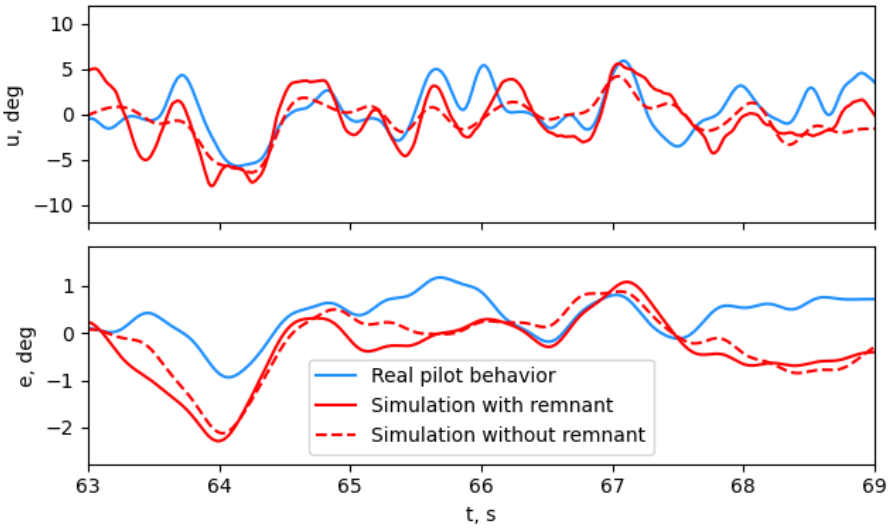


Figure 4.22: An example of simulated pilot behavior versus real pilot behavior in the same tracking task. (Time traces from participant 1, run 100)

were used to generate the remnant signal, so that a fair comparison between the different remnant gains could be made. Each setting was tested five times to account for the random in training the network.

The outcome of the data augmentation experiment is shown in Figure 4.23. An unexpected result was found from this experiment. Namely, it appears that increasing the magnitude of the remnant (of the simulated pilot behavior that is used as training data), increases the validation accuracy on real pilot time traces. This may indicate that the estimated filter gains K from Section 4.3.1 are too low, but further research will have to be performed to make any conclusive remarks on these findings. The fact that a validation accuracy of up to 76% was reached is a promising prospect for the proposed data augmentation method.

It was also tested whether the inclusion of remnant in the pilot model induced any bias into the predictions of the neural network model. This was investigated by recording the percentage of pilot time trace samples that networks, trained with different remnant magnitudes, classified as either unskilled or skilled. The results of this exercise are summarized in Figure 4.24, here the total height of each bar indicates the percentage of samples that the trained network put in each class (i.e. the bias). Within each bar the percentage of correctly classified samples is shown (i.e. accuracy). There is a small discrepancy between the validation accuracy shown in Figure 4.23 and in Figure 4.24. This is because the accuracy in Figure 4.23 is the maximum encountered validation accuracy during training, whereas the accuracy in Figure 4.24 is the validation accuracy per class when employing trained network models.

From these results, no clear relation between remnant gain and network prediction bias could be derived. Overall, the network trained with generated data appears to be more biased towards classifying samples as 'skilled'. Only when the remnant gain is zero (i.e. there is no remnant), does the network have a bias towards 'unskilled'. Strangely, the largest bias is observed when the remnant gain is set to one, even though this setting was designed to be the most accurate representation of the real pilot data. It can also be observed that the classification accuracy is lowest for the class that the network is biased towards. This makes sense, since bias naturally triggers more false positive predictions.

Further research will have to be conducted to be able to explain the observed training behavior. Each setting should be tested more often to get results that are more meaningful statistically. The explainability methods, that will be introduced in Section 4.4, may be helpful tools to understand how different training settings affect the network's decision making.

4.3.3. Takeaways

This section discussed how a quasi-linear pilot model was utilized to generate additional training data. The pilot model consists of linear transfer functions and a remnant that is modeled as low-pass filtered

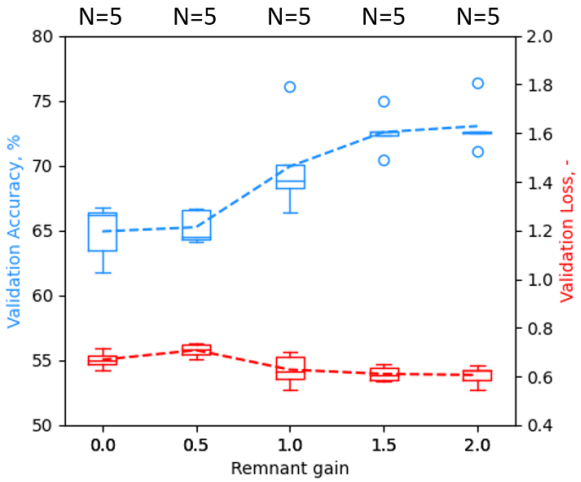


Figure 4.23: Validation accuracy and validation loss when training data = simulated, and validation data = real pilot behavior. Different remnant gains were used to simulate the training data.

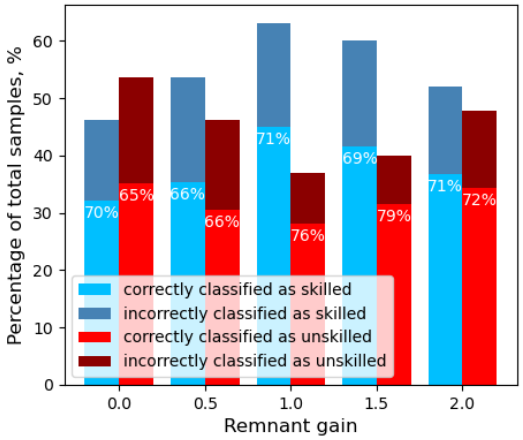


Figure 4.24: Neural networks trained with different levels of remnant gain show different bias in their class predictions.

white noise. For every recorded tracking run of the actual participants, a simulated counterpart was generated.

To test the feasibility of this data augmentation method, the LSTM model was trained using solely generated data, after which it was validated on the actual pilot data. From this analysis it was found that the LSTM model trained with simulated data achieves about 70% validation accuracy on real pilot data, on average (compared to approximately 80% when training on real pilot data). Removing the remnant from the pilot simulations lowers this validation accuracy to about 65%, whereas doubling the magnitude of the remnant increases the validation accuracy to approximately 73%. There is a significant uncertainty in the trained model performance (i.e. between 66% and 76% with the standard remnant setting), the cause of this variability will have to be further investigated.

It was also found that training the LSTM model on simulated data will generally lead to a classification model that is biased towards classifying real pilot samples as 'skilled'. There appears to be no clear correlation between remnant gain and model bias. More research will have to be done to make conclusive remarks about the above observations.

4.4. Explainable Artificial Intelligence Implementation

In this section a preliminary implementation of eXplainable Artificial Intelligence (XAI) will be discussed. Theoretical background about explainability in machine learning has been provided in Section 3.4. From the observed literature, it was concluded that a model-agnostic explainability method is most suitable for this preliminary experiment.

In Section 4.4.1 a method is described that calculates the relative importance of each input variable, Section 4.4.2 extends this method to generate class activation maps.

4.4.1. Feature Importance

As has been explained in Section 3.4, there are methods that quantify feature importance. This form of XAI calculates the contribution of each input that led to a certain output. In this preliminary experiment, a model-agnostic method called SHAP (Lundberg & Lee, 2017) was used to estimate the relative contribution of the input variables. This method was implemented by using the `SHAP library` for `Python`.

Specifically, the `GradientExplainer` function was used to find local estimates of marginal feature contributions to model output. This function uses *expected gradients*, which is an extension of the *integrated gradients* method by Sundararajan, Taly, and Yan (2017). The integrated gradients method compares some baseline input x' to an actual input x. A path is drawn between x' and x, after which gradients are computed at each point along the path. Integrated gradients are found by summing up the individual gradients (Sundararajan et al., 2017). The integrated gradients indicate the relative con-

tribution of every dimension in x (i.e. every input variable).

The marginal contribution of the input features can be calculated for a single model prediction (local explanation), an example of this was shown in Figure 3.24. By calculating the feature contribution for every instance in the data set, the contributions can be summed to find the overall feature importance of the trained model (global explanation). Note that the relative contribution of the input variables are different for every (locally explained) model prediction, i.e. for every classified sample a different input can be more important than another. Therefore, by calculating the importance of the input variables for every data point, an average feature importance can be estimated.

Precisely the method described above was used to estimate the global feature importance of the trained LSTM model. The results of this exercise are shown in Figure 4.25, here the blue bars indicate the relative feature importance for the LSTM model trained with real human pilot data, whereas the red bars show feature importance for the same model trained with simulated pilot data (remnant gain equal to one). For both these differently trained models, the feature importance was computed on class predictions of *real* pilot data. The settings were kept as they have been presented in Table 4.1 and Section 4.2.2, with *experience* based labeling (first and last fifteen runs).

What is interesting to see in Figure 4.25, is that for both models (either trained with *real* or *simulated* data) the tracking error e and the time derivative of control output $\dot{u}$ are the two most important features to classify samples. This indicates that, according to the trained models, experienced pilots predominantly distinguish themselves from inexperienced pilots by time traces of the magnitude of tracking error and the speed with which they operate the control column. Also note that, for both models, the time derivative of the tracking error $\dot{e}$ appears to be the least influential feature.

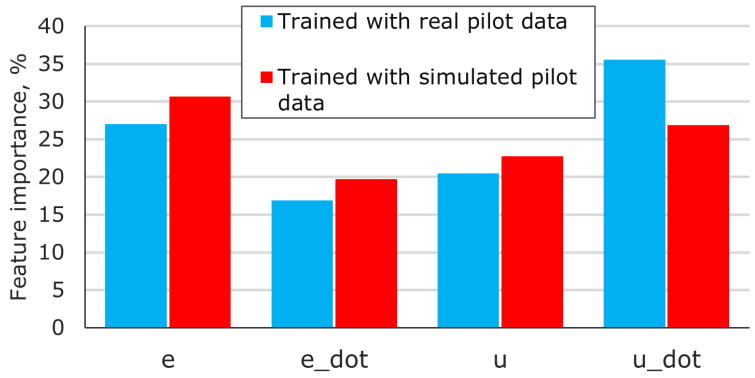


Figure 4.25: Global feature importance, computed with SHAP, for two differently trained LSTM models.

To provide more context to the earlier presented feature importance, consider Figure 4.26. This graph displays the correlation between input variable variance and class prediction of the model trained with real pilot data. Every dot is an entire run, the color of the dot is the average predicted class of all samples from that run, the horizontal position of the dot indicates the variance of the respective feature over that tracking run, the vertical position is only to separate dots in densely populated regions. Lastly, the overlaying bars indicate the average classification over the respective bin width (i.e. it can be interpreted as the average color in that area).

From Figure 4.26 it can be seen that tracking runs with low tracking error variance will, on average, be classified as skilled (this correlation is also observable in Figure 4.14). For the other input variables the opposite is true: low variance is related to 'unskilled' pilot predictions. Figure 4.26 also reinforces the findings from Figure 4.25, as it can be seen that σ_e^2 and $\sigma_{\dot{u}}^2$ show the strongest correlation with model output. This can be recognized by the decisive magnitude of the bar charts that is either close to zero or close to 100. This is unlike the other inputs where the color is more diluted and the bar charts hover around 50%, especially $\sigma_{\dot{e}}^2$ appears to be weakly correlated to model output (as was already confirmed by Figure 4.25).

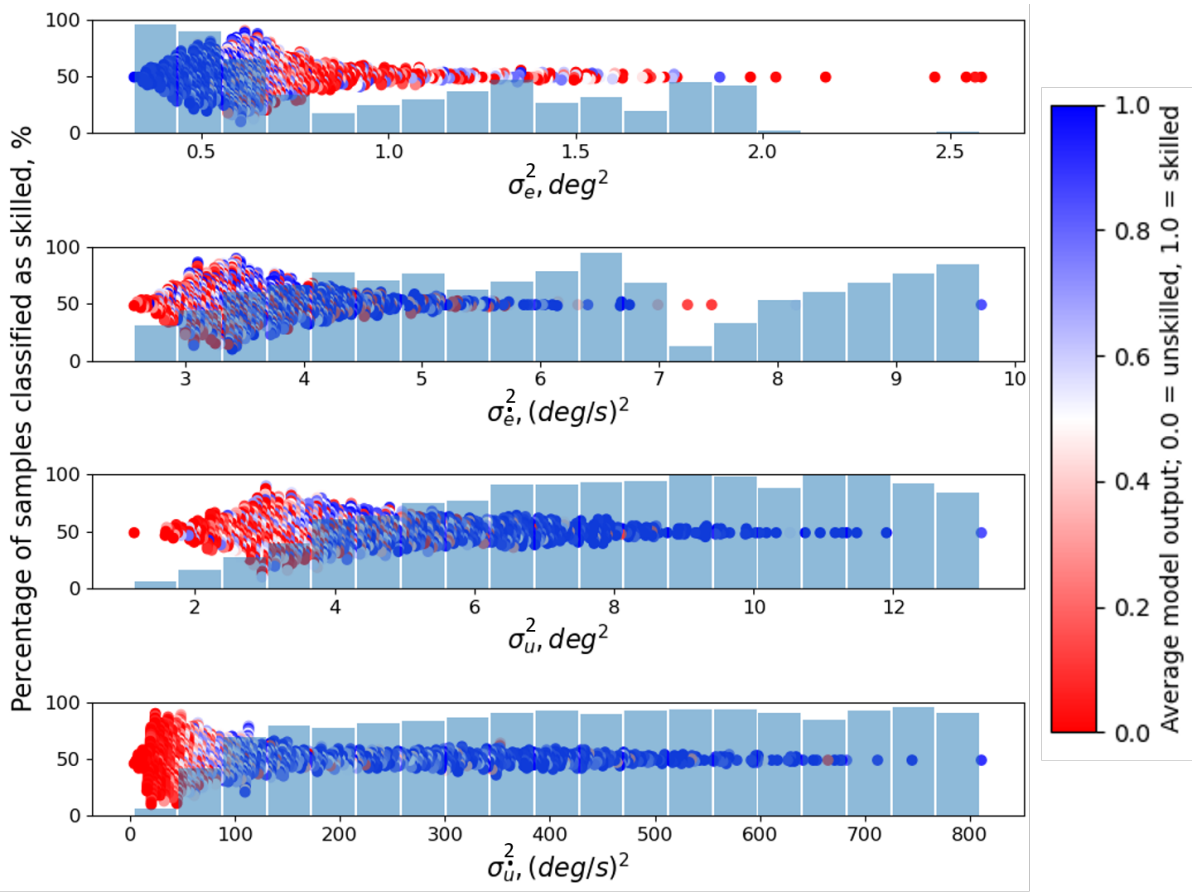


Figure 4.26: Visible correlation between variance of input variables and average model classification output.

4.4.2. Class Activation Map

Another potential use case for local feature importance explanation of model output is to generate Class Activation Maps (CAMs). These maps, as has been discussed in Section 3.4, indicate discriminative regions in the time traces that lead to certain model outputs. In Section 3.4 CAM was described as a method that can only be utilized for specific CNN models. This is true, however, an *estimated* CAM can be computed for RNNs using feature importance values. The following paragraphs will describe how this is done and explain why it is inferior to actual CAMs generated with CNNs.

CAM from feature importance: The marginal contribution of each input variable that lead to a specific model output can directly be utilized to create a graph like the one shown in Figure 4.27a. Here it can be seen that for every sample (1.6 seconds) the trained LSTM model outputs a predicted class (seen in the bottom sub-graph labeled 'class'). Consecutively, the highlighted areas of the four other sub-graphs indicate the relative contribution of each input variable to reach the class prediction. If a section of a time trace is red, then it means that that region pushed the model prediction towards 'unskilled', contrarily blue areas indicate pilot behavior that leads to 'skilled' classification. The intensity of the color indicates that magnitude of contribution. Note that individual contributions of separate features can contradict each other.

The y-axes of the feature time traces in Figure 4.27 have no units, this is because these input features are *standardized* (as they are presented to the model). Specifically a run halfway in the pilot training process was chosen to generate Figure 4.27, i.e. run number 50 out of 100. This is because the trained LSTM model is most uncertain for runs around this index, bringing out discriminative regions in the time traces more predominantly. Runs around index 1 or 100, for example, will simply be entirely red or blue, respectively.

The downside of Figure 4.27a is that the highlighted areas stretch over an entire sample, making it difficult to identify the precise discriminative region that led to the model output. To overcome this

problem, a moving average method is proposed. By using a sliding window with a specified percentage of overlap between consecutive classified samples, the model output (and associated feature importance) can be averaged. To illustrate this, consider Figure 4.27b, in this figure a 50% overlap between consecutive samples is used. Notice that this doubles the resolution with which discriminative regions can be indicated. This resolution can be further increased by, for example, using 90% overlap as is shown in Figure 4.27c.

Although the presented method is able to effectively capture specific regions of time traces that lead to class predictions, there are two major downsides to the proposed method. 1) The indicated relative contributions are *estimates*, meaning that these numbers do not fully accurately describe the model's behavior. This is not the case for CAM with CNNs, here an actual look into the model's decision making is provided. 2) The proposed method takes longer to compute than CNN CAMs. Estimating the local feature importance for a single prediction is already computationally expensive, using 90% overlap makes it even more time consuming to produce these figures.

Observations from CAM: From Figure 4.27c some preliminary observation can be made. For example, it appears that if e moves away from zero (e.g. at 20.5s) this leads to 'unskilled' predictions. On the contrary, if e moves back towards zero (e.g. at 22.5s and at 25.5s) the output is pushed towards 'skilled'. This pattern is reinforced by $\dot{e}$, which essentially indicates the direction e is moving to (i.e. moving down when e is negative is red, whereas moving down when e is positive is blue). For $\dot{u}$ it seems that homogeneous 'sine-like' behavior portrays unskilled behavior, and vice-versa. There seems to be little to no influence at all by u.

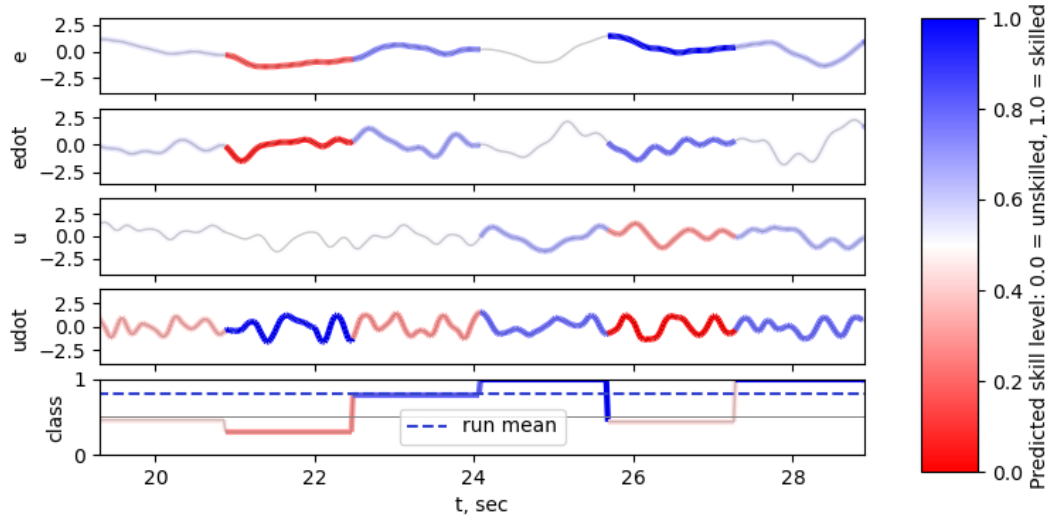
It must be noted that the presented observations are based on only a very small portion of the data. The presented XAI method will have to be utilized more extensively to observe more AI interpretations of pilot behavior. Additionally, the found results will have to be investigated more systematically and thoroughly. For example, certain reoccurring patterns that lead to class activation could be collected and compared. This may give insight into what 'shapes' in time traces of pilot behavior belong to trained or untrained behavior. Additionally, side by side comparisons of CAMs could be done between differently trained neural network models, or between differently classified participants performing the same task.

4.4.3. Takeaways

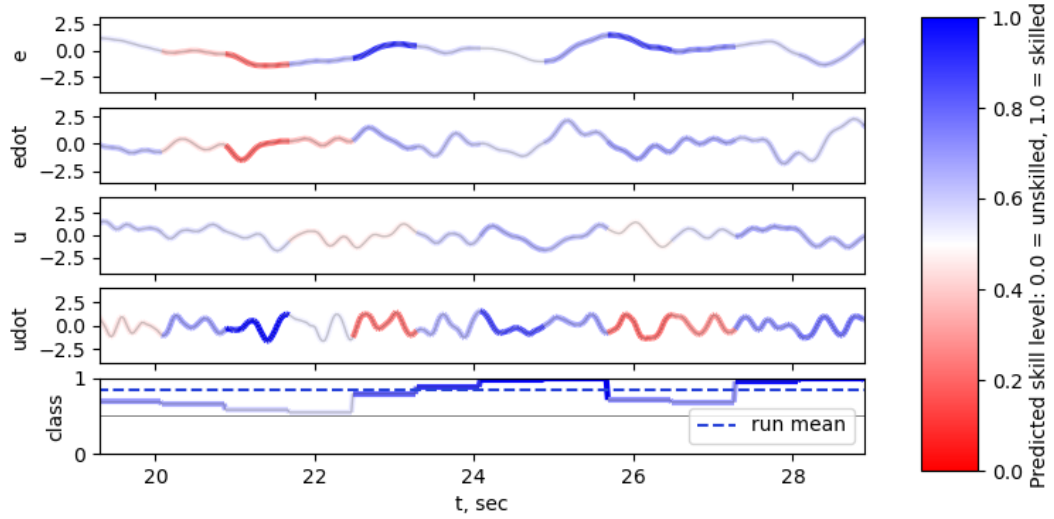
This section detailed the implementation of eXplainable Artificial Intelligence to the preliminary LSTM model. A model-agnostic post hoc explainability method called SHAP was used to calculate the relative contribution of each input variable to a certain model prediction.

By calculating the relative contribution of the input variables for every sample in the data, a total feature importance can be determined. From this analysis it was found that, globally, the tracking error e and the time derivative of the control output $\dot{u}$ are the most important features for the LSTM model to classify time traces.

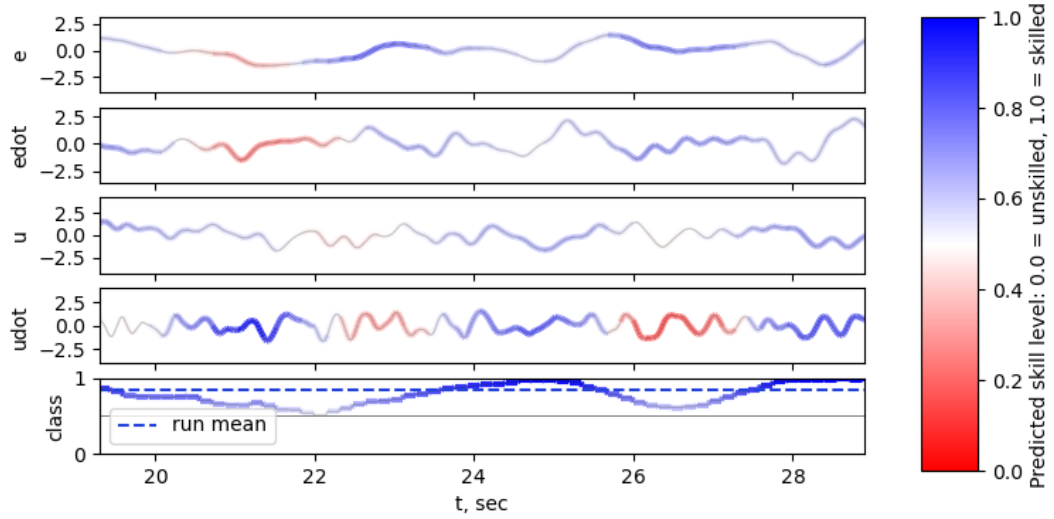
Local feature importance calculations can be used to create class activation maps. This allows the identification of discriminative regions in the time traces that lead to certain model output. This visualization tool gives some more insight into the model's decision making, but it will have to be tested more extensively before any meaningful conclusions can be drawn.



(a) Class activation map with 0% overlap between classified samples.



(b) Class activation map with 50% overlap between classified samples.



(c) Class activation map with 90% overlap between classified samples.

Figure 4.27: Class activation map for trained LSTM model, made using SHAP library for `Python`. (Time traces from participant 1, run 50)

4.5. Results and Discussion of Preliminary Experiment

The preliminary experiment investigated three topics: 1) classifying pilot skill level with artificial neural networks, 2) generating additional data with a cybernetic pilot model, and 3) interpreting the output of the trained neural network model. The following sections will briefly discuss the preliminary results for all three of these topics.

4.5.1. Classifying Pilot Skill Level with LSTM Model

From the preliminary experiment it was found that, without any hyperparameter optimization, the trained LSTM model can correctly classify trained and untrained pilots with about 81% accuracy. Ideally, of course, this classification would be done with 100% accuracy. The following four reasons may be the cause of the LSTM model falling short in performance.

- The hyperparameters need to be optimized.
- A different neural network architecture must be employed.
- Not every training sample is representative of its respective class label.
- There is too little training data.

A brief sensitivity analysis indicated that there may be room for improvement through **hyperparameter optimization**. Testing the network extensively, under different (combinations of) settings, could identify more optimal hyperparameter values. This could significantly increase the classification performance.

It may be the case that the selected **neural network architecture (LSTM) is a suboptimal option** for this thesis. A completely different architecture (i.e. CNN) will have to be tested, this will allow comparison of performance between the different models.

It could be that, with the selected labeling method, **not every sample is a good representation of the class it is labeled as**. Two observations support this claim: 1) using a performance based label increased validation accuracy, and 2) there is a significant difference in achieved validation accuracy when using different selections of train/test data.

Some samples may be poorly labeled due to time-variant behavior of the participants, i.e. some samples of a 'skilled' subject may actually be unskilled behavior (and vice-versa). It could also be due the difference between participants, i.e. different subjects may utilize different control strategies or some may have 'talent' and be quicker to learn how to exert skilled behavior. This all leads to a 'noise' of sorts in the training (and validation) data, making it more difficult for the network the generalize. A potential method to further investigate the nature of this problem is 'cross-validation'.

Lastly, it may be the case that there is **too little training data** to effectively train the neural network. From the sensitivity analysis it was found that lowering the overlap between samples (and thus lowering that amount of training samples) severely decreased the validation accuracy. This could be an indication that the presented results are suboptimal due to scarce training data. This will have to be further investigated by recording the impact of purposely leaving out a portion of the training data. A potential solution to this problem may lie in the proposed data augmentation method.

4.5.2. Data Augmentation with Quasi-Linear Pilot Model:

The preliminary data augmentation experiment found that training the LSTM model with simulated pilot data, results in a classification accuracy of real pilot data between 66% and 76%.

Although these are quite promising results (considering that the network is trained with *only* simulated data), this is not an entirely fair representation of the effectiveness of the proposed data augmentation method. Namely, the simulated training data are based on real tracking runs that are also used as validation data. This means that, in a way, there is overlap between the training and testing data, resulting in inflated test results.

What also stands out is the wide spread of validation accuracy (i.e. 10%) found during this experiment. Performing more training runs could produce results that are statistically more meaningful than the results presented in this preliminary phase.

From the conducted analysis, it was found that the amount of remnant in the simulated data strongly influences the training performance. The preliminary results indicate that doubling the amount of remnant, averagely increases the validation accuracy by three percent. This could be an indication that the remnant model has to be improved. A potential improvement would be to calculate more specific remnant parameters for each simulated individual run of every participant.

Lastly, it must also be noted that there were some instabilities when simulating the pilot behavior. For certain runs of some participants, the pilot model parameters (taken from Pool et al. (2016)) had extremely high values. This resulted in the `Simulink` model crashing due to infinite derivatives. A temporary solution was to overwrite these extremely high parameter values with averages. As this was only the case for seven out of all tracking runs, it is expected to have had very little effect on the presented results.

4.5.3. Explaining the Trained LSTM Model:

The last topic, that was investigated during this preliminary phase, is explainable artificial intelligence. Using *expected gradients* an estimated feature importance could be produced to interpret the trained LSTM model. From this analysis it was found that the tracking error e and the control output speed $\dot{u}$ are the most important input variables. Interestingly, this was also the case for the LSTM model trained with simulated pilot data.

It should be noted, however, that the calculated feature contributions are *estimates*. This is more problematic for local explanations than it is for global explanations, since the global estimates average over a large set of predictions.

The gradient explanation method was also utilized to generate class activation maps. This is a visual tool that highlights the discriminative area in the time traces that lead to a certain class prediction. Although effective, the presented method is computationally expensive and based on estimates. CNNs could potentially solve both these problems, lowering computation time and providing direct insight into the model's decision making.

The class activation maps may also be extended to provide global explanation. For example, specific shapes or patterns that consistently lead to a certain class prediction could be identified.

5 Research Plan

The goal of this preliminary phase was to shed some light on topics relevant to this thesis. Simultaneously, (partial) answers to some of the posed research questions have been established from both literature and preliminary experimenting. To ensure that all the remaining research questions will be answered by the end of the main phase of this thesis, a research plan has to be composed.

First the research questions will be repeated, after which the remaining unanswered questions are identified. A project planning is proposed to find the answers to the remaining questions.

5.1. Research Questions

The introduction of this report introduced the following main research question: **how can deep learning be used to identify pilot skill level?** In order to answer this question in completeness, a list of sub-questions was introduced. The remaining questions are the following (the crossed out text indicate the posed sub-questions that have been answered by the findings presented in this preliminary report):

Research sub-questions

1. **How can artificial intelligence be used to classify time series data?**
   (a) ~~What algorithms can be used to perform this task?~~
   (b) How must the training data be structured?
   (c) What influence do hyperparameters have on the classification performance?
   (d) What is the optimal performance achieved with the proposed method?
2. **How can pilot modeling be used to generate additional training data?**
   (a) ~~What type of model can be used to simulate pilot behavior?~~
   (b) What influence do pilot model parameters have on classification performance?
   (c) What is the added benefit of the proposed data generation method?
3. **How can an explainability component be added to the classification model?**
   (a) ~~What explainability methods are available?~~
   (b) What input parameters influence the classification model's output and how?
   (c) How can the proposed explainability method be used to interpret the classification model's decision making?

5.2. Project Planning

In order to effectively plan the main stage of this thesis, the remainder of the research is split into three phases. Each phase will aim to answer one of the three sub-questions. The following sections will describe precisely what actions will be undertaken in each phase to come to the final answers of the research questions.

5.2.1. Phase I: Optimize Deep Network Classifier

The conducted literature review provided a list of potential algorithms that may be used to classify time series, answering question **1(a)**. However, only by actually putting the suggested networks to practice, can a comparison in performance be accomplished. Therefore, the main phase of this thesis shall implement a state-of-the-art CNN network, so that its performance can be compared to the stacked LSTM model used in preliminary experiment.

As has been mentioned in the discussion of the preliminary results, the classification performance of the network is highly dependent on the class labeling method. The preliminary experiment helped to identify some potential labeling methods, but more thorough research has to be done to select an optimal method. A potential tool that can be used to more efficiently recognize poorly labeled samples is cross-validation. The above will help to answer question **1(b)**.

Once an optimal network architecture has been selected, and a final labeling and structuring is applied to the training data, the hyperparameters may be optimized. Optimizing the hyperparameters will provide optimal classification performance and will simultaneously identify where the sensitivity of the network lies. This will answer questions **1(c)** and **1(d)**.

To summarize, the following steps will be taken during phase I. This phase is expected to take a total of six weeks.

- Implement state-of-the-art TSC CNN network (e.g. FCN, ResNet (Wang et al., 2017), or InceptionTime (Fawaz et al., 2020)) and compare classification performance.
- Optimize labeling method.
- Perform cross-validation to identify potential underfitting/overfitting.
- Optimize model performance with hyperparameter optimization.

5.2.2. Phase II: Implement Final Data Augmentation Model

The preliminary phase of this thesis proposed a quasi-linear pilot model that may be used to generate additional training data for the machine learning algorithm. The preliminary experiment indicated that the simulated data is accurate enough to teach a neural network to classify real pilot data. However, an effort should be made to increase the accuracy of the simulated data.

A potential improvement lies in the remnant model. The preliminary research has proven that the remnant strongly influences the classification performance. The exact nature of this correlation should be further discovered to optimize the remnant settings. Testing and improving the pilot model shall answer question **2(b)** and result in optimized simulation data.

Once the pilot model is optimized, it should be investigated exactly how useful the proposed data augmentation method is. This will be tested in a case study. The case study will quantify the benefit of data augmentation with cybernetic pilot modeling when there is little training data, answering question **2(c)**. This may also help identify whether there is too little training data under the current settings, and if this can be overcome.

Phase II of the main stage is expected to take four weeks. The following lists summarizes the steps that will be taken during this phase.

- Improve pilot model to remove bias and increase accuracy (when training on simulated data and testing on real pilot data).

- Quantify added benefit when used as data augmentation method in case study.

5.2.3. Phase III: Utilize Explainable Artificial Intelligence

Out of the three pillars of this thesis, the XAI part is the most developed. This is because, in principle, it is already fully functional and can be deployed in its current state.

Two potential problems with the current XAI method is that it *estimates* model decision making, and that it is computationally expensive. These two problems may be overcome with a CNN architecture, however, this is completely dependent on whether the CNN outperforms the LSTM in terms of accuracy.

Regardless of what neural network architecture is selected, XAI will be used to interpret the results of phase I and phase II. This interpretation through XAI will answer questions **3(b)** and **3(c)**.

This phase is expected to take no more than three weeks. The following two items summarize the activities of phase III.

- Compare and explain optimized deep learning models (e.g. in terms of variable importance).
- Extract insightful information from tool, e.g. specific patterns that belong to unskilled/skilled behavior.

6

Conclusion

In conclusion, this preliminary report documented the findings of a combined *literature study* and *preliminary experiment*. The goal of this preliminary phase was to get accustomed with deep learning, pilot modeling, and explainable artificial intelligence, so that these topics could effectively be combined in the *main phase* of this thesis.

Ultimately, the goal of this thesis is to utilize deep learning as a tool to effectively classify pilot skill level. This goal will be achieved by answering all of the posed research questions. The following paragraphs will conclude how each phase of this thesis has contributed to reaching the research objective.

Literature study: The literature study highlighted the current state-of-the-art and identified where this thesis could contribute to the existing body of knowledge. Namely, this thesis is the first to utilize deep learning to classify pilot skill level based on control output time traces, without the use of additional sensors (e.g. eye trackers or heart rate monitors). Additionally, it is also the first to attempt to perform data augmentation using cybernetic pilot modeling.

The first step taken was to identify a potential source of training data that could be used to teach a neural network to identify pilot skill level. It was concluded that the used data must have examples of both untrained/trained individuals performing the same task. Simultaneously, it was reviewed how the training data could be augmented using a quasi-linear pilot model. This partially answered question **1(b)** and identified a solution to question **2(a)**.

An extensive review of artificial neural network literature helped to understand the working principle of this class of machine learning algorithms. The foundation of knowledge was then applied to review the application of neural networks in the domain of time series classification. From this investigation it was found that recurrent neural networks and convolutional neural networks are the most promising type of network for the goal of this thesis, answering question **1(a)**.

Lastly, the current trends in the field of explainable artificial intelligence were reviewed. This analysis led to a selection of methods that were deemed applicable for this thesis. It was found that only *post-hoc explainability* could be used to interpret the selected deep learning networks, resolving question **3(a)**.

Preliminary experiment: During the preliminary experiment some of the found literature was put to the test. A stacked LSTM model was employed to perform time series classification on the collected pilot data, this provided a preliminary perspective on questions **1(b)**, **1(c)**, and **1(d)**. Namely, it was found that labeling the training data based on the *experience* of the participants led to a fairer comparison than when the data was labeled based on an assumed *performance* measure. Currently, the trained LSTM model achieves roughly 81% accuracy in classifying pilots as skilled or unskilled. A brief sensitivity analysis concluded that there is room for improvement through hyperparameter optimization.

Training the same LSTM model (under the same settings) with only simulated pilot data, resulted classification accuracy up to 76%. However, the performance was very inconsistent and must thus be further reviewed to answer **2(b)** and **2(c)**.

The preliminary implementation of the explainable artificial intelligence method helped to interpret the results of the preliminary experiment. An expected gradients method was used to estimate the rel-

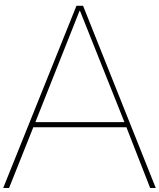
ative importance of the input variables. The feature contribution estimates were also used to generate class activation maps. This gave an indication of how question **3(b)** and **3(c)** may be answered.

Main phase planning: During the main phase of this research, complete answers to all the posed research questions will be provided. In order to achieve this, the work is split into three phases: I) optimization of deep network classifier, II) implementation of final data augmentation model, and III) utilization of explainable artificial intelligence.

Phase I is estimated to be the most time consuming phase. In this phase, different neural network architectures will be compared so that an optimal network type can be selected. Once the structure of the neural network is established, optimization will take place to achieve the highest possible classification performance. This will answer questions **1(b)**, **1(c)**, and **1(d)**.

With the optimized neural network model ready for deployment, phase II will conduct a study to test the effectiveness of using a cybernetic pilot model to simulate additional training data. Before quantifying the added benefit, the pilot model must first be reviewed and improved. This implementation will provide answers to questions **2(b)** and **2(c)**.

Lastly, phase III will answer questions **3(b)** and **3(c)** by utilizing the proposed explainable artificial intelligence method to extract insightful information from differently trained classification models.



Graphs of Activation Functions

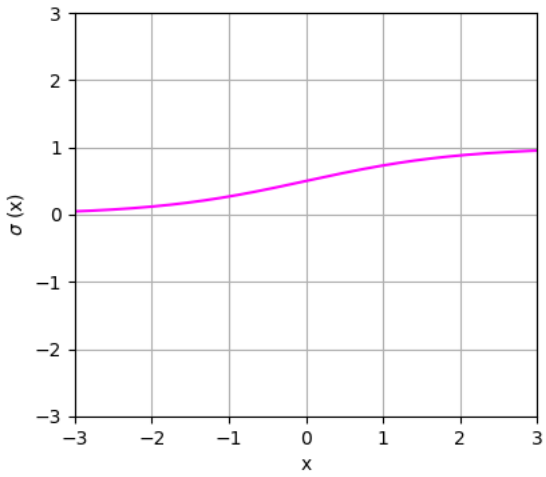


Figure A.1: Sigmoid function squishes values between zero and one

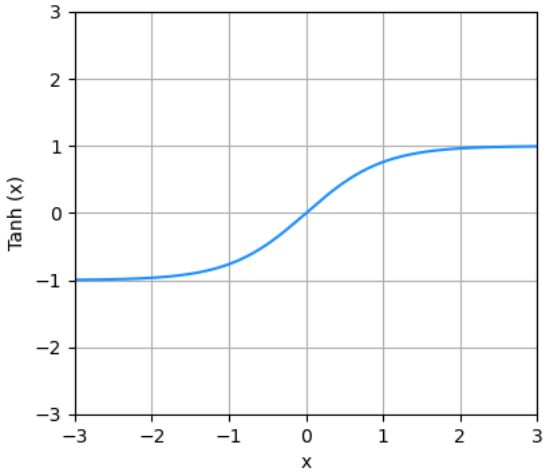


Figure A.2: Tanh function squishes values between minus one and one

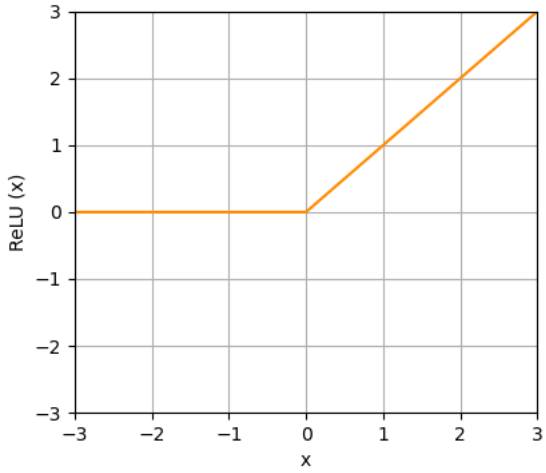


Figure A.3: ReLU function is given by taking the maximum value between zero and x

B

Additional Figures

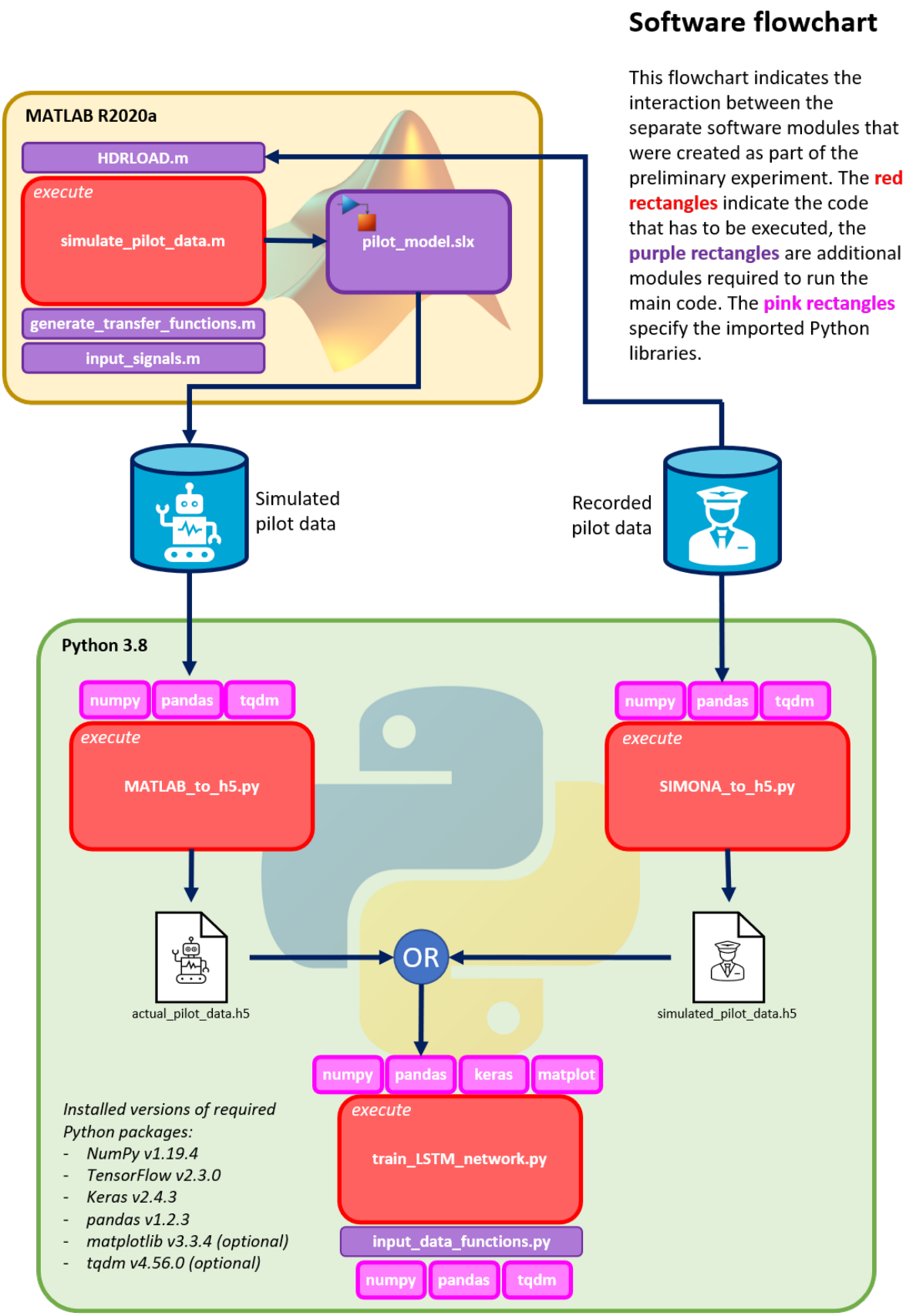


Figure B.1: Software flowchart indicating the interaction between all software modules (and libraries) used for the preliminary experiment.

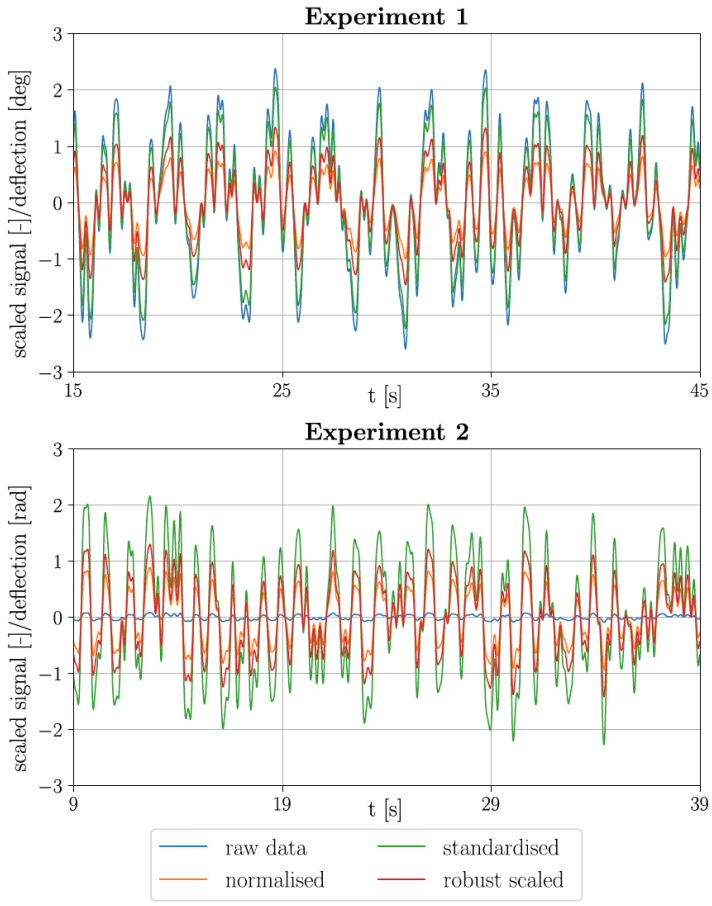


Figure B.2: Example of scaling methods applied to two different datasets. Image taken from (Versteeg, 2019)

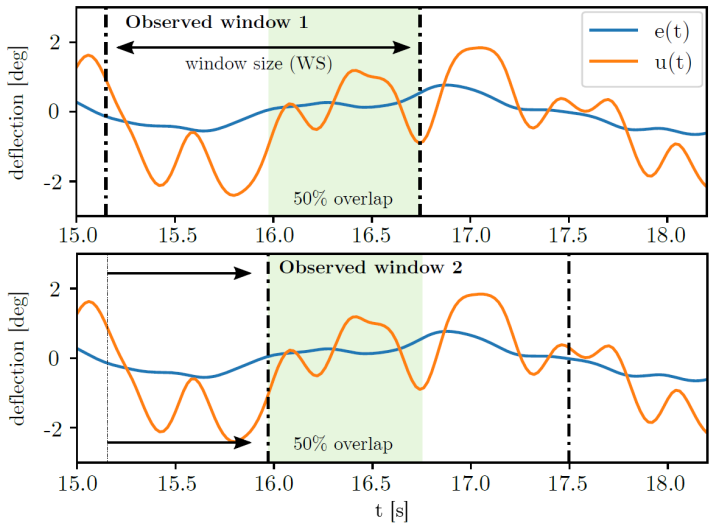


Figure B.3: Example of sliding window with 50% overlap. Image taken from (Versteeg, 2019)

C

Additional Functions

Time derivative estimates
The following numerical differentiation methods were used to compute the time derivatives of the pilot data.

$$\hat{f}_{t=i}^{(1)} = \frac{f\left(x_{i+1}\right) - f\left(x_{i-1}\right)}{2\Delta t} + \mathcal{O}\left(\Delta t^2\right) \quad \text{central difference} \tag{C.1}$$

$$\hat{f}_{t=0}^{(1)} = \frac{f\left(x_{i+1}\right) - f\left(x_i\right)}{\Delta t} + \mathcal{O}(\Delta t) \quad \text{one-sided forward difference} \tag{C.2}$$

$$\hat{f}_{t=t}^{(1)} = \frac{f\left(x_i\right) - f\left(x_{i-1}\right)}{\Delta t} + \mathcal{O}(\Delta t) \quad \text{one-sided backward difference} \tag{C.3}$$

Normalizing
A time trace sequence $z\,(t)$ can be scaled between -1 and 1 to ensure positive and negative sign convention using Eq. (C.4).

$$\hat{z}\,(t) = \frac{z\,(t)}{\max\left(|\min\left(\vec{z}\right)|, |\max\left(\vec{z}\right)|\right)} \tag{C.4}$$

Standardizing
In Eq. (C.5) a time trace sequence $z\,(t)$ is standardized by removing the mean μ_z and dividing over the standard deviation σ_z.

$$\hat{z}\,(t) = \frac{z\,(t) - \mu_z}{\sigma_z} \tag{C.5}$$

Learning curve model
This learning curve model was used in the research of Pool et al. (2016). The learning curve model (Eq. (C.6)) has three parameters: the initial value p_0, the asymptotic value p_a, and the learning rate F (Levison, Lancraft, & Junker, 1979).

$$y_{\mathrm{lc}}(x) = p_a + (1 - F)^x\,(p_0 - p_a) \tag{C.6}$$

Bibliography

Acharya, U. R., Oh, S. L., Hagiwara, Y., Tan, J. H., Adam, M., Gertych, A., & Tan, R. S. (2017). A deep convolutional neural network model to classify heartbeats. *Computers in Biology and Medicine*, *89*, 389–396. https://doi.org/10.1016/j.compbiomed.2017.08.022

Al-Qizwini, M., Barjasteh, I., Al-Qassab, H., & Radha, H. (2017). Deep learning algorithm for autonomous driving using googlenet. *2017 IEEE Intelligent Vehicles Symposium (IV)*, 89–96. https://doi.org/10.1109/IVS.2017.7995703

Arras, L., Montavon, G., Müller, K.-R., & Samek, W. (2017). *Explaining recurrent neural network predictions in sentiment analysis*. arXiv: 1706.07206 [cs.CL].

Arrieta, A. B., Díaz-Rodríguez, N., Ser, J. D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2019). *Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai*. arXiv: 1910.10045 [cs.AI].

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 153–160.

Berrar, D. (2019). Cross-validation. In S. Ranganathan, M. Gribskov, K. Nakai, & C. Schönbach (Eds.), *Encyclopedia of bioinformatics and computational biology* (pp. 542–545). Academic Press. https://doi.org/10.1016/B978-0-12-809633-8.20349-X

Bottou, L. (2012). Stochastic gradient descent tricks. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade: Second edition* (pp. 421–436). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_25

Bradbury, J., Merity, S., Xiong, C., & Socher, R. (2016). *Quasi-recurrent neural networks*. arXiv: 1611.01576 [cs.NE].

Castelvecchi, D. (2016). Can we open the black box of ai? *Nature*, *538*, 20–23. https://doi.org/10.1038/538020a

Cortez, P., Teixeira, J., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Using data mining for wine quality assessment, 66–79. https://doi.org/10.1007/978-3-642-04747-3_8

Cunningham, P., & Delany, S. (2007). K-nearest neighbour classifiers. *Mult Classif Syst*.

Donahue, J., Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2014). Long-term recurrent convolutional networks for visual recognition and description. *Arxiv*, *PP*. https://doi.org/10.1109/TPAMI.2016.2599174

Došilović, F. K., Brčić, M., & Hlupić, N. (2018). Explainable artificial intelligence: A survey. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 210–215. https://doi.org/10.23919/MIPRO.2018.8400040

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(61), 2121–2159.

Eickenberg, M., Gramfort, A., Varoquaux, G., & Thirion, B. (2017). Seeing it all: Convolutional network layers map the function of the human visual system. *NeuroImage*, *152*, 184–194. https://doi.org/https://doi.org/10.1016/j.neuroimage.2016.10.001

Erhan, D., Courville, A., Bengio, Y., & Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 201–208). JMLR Workshop; Conference Proceedings.

Esteban, C., Hyland, S. L., & Rätsch, G. (2017). *Real-valued (medical) time series generation with recurrent conditional gans*. arXiv: 1706.02633 [stat.ML].

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*(7639), 115–118. https://doi.org/10.1038/nature21056

Evgeniou, T., & Pontil, M. (2001). Support vector machines: Theory and applications. *2049*, 249–257. https://doi.org/10.1007/3-540-44673-7_12

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2018). Data augmentation using synthetic data for time series classification with deep residual networks. *ArXiv*, *abs/1808.02455*.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963. https://doi.org/10.1007/s10618-019-00619-1

Fawaz, H. I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., & Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962. https://doi.org/10.1007/s10618-020-00710-y

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669. https://doi.org/https://doi.org/10.1016/j.ejor.2017.11.054

Gao, Y., Hendricks, L. A., Kuchenbecker, K. J., & Darrell, T. (2016). *Deep learning for tactile understanding from visual and haptic data*. arXiv: 1511.06065 [cs.RO].

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Hall, P. (2020). *On the art and science of machine learning explanations*. arXiv: 1810.02909 [stat.ML].

Hannun, A. Y., Rajpurkar, P., Haghpanahi, M., Tison, G. H., Bourn, C., Turakhia, M. P., & Ng, A. Y. (2019). Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine*, *25*(1), 65–69. https://doi.org/10.1038/s41591-018-0268-3

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition*. arXiv: 1512.03385 [cs.CV].

Hermans, M., & Schrauwen, B. (2013). Training and analyzing deep recurrent neural networks. *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, 190–198.

Hess, A. R., & Peng, C. (2018). Design for robust aircraft flight control. *Journal of Aircraft*, *55, No. 2*.

Hess, R. A. (1995). A model-based analysis of handling qualities and adverse aircraft-pilot coupling in high angle of attack flight. *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*, *3*, 2663–2669. https://doi.org/10.1109/ICSMC.1995.538185

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2018). *Gans trained by a two time-scale update rule converge to a local nash equilibrium*. arXiv: 1706.08500 [cs.LG].

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735

Hosman, R. J. A. W., & Stassen, H. (1999). Pilot's perception in the control of aircraft motions. *Control Engineering Practice*, *7*(11), 1421–1428. https://doi.org/https://doi.org/10.1016/S0967-0661(99)00111-2

Hosman, R. J. A. W., & van der Vaart, J. C. (1978). *Vestibular models and thresholds of motion perception. results of tests in a flight simulator* (Internal Report LR-265). Delft University of Technology, Faculty of Aerospace Engineering. http://repository.tudelft.nl

Jain, A., Singh, A., Koppula, H. S., Soh, S., & Saxena, A. (2016). Recurrent neural networks for driver activity anticipation via sensory-fusion architecture. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 3118–3125. https://doi.org/10.1109/ICRA.2016.7487478

Jiang, W., & Yin, Z. (2015). Human activity recognition using wearable sensors by deep convolutional neural networks. *Proceedings of the 23rd ACM International Conference on Multimedia*, 1307–1310. https://doi.org/10.1145/2733373.2806333

Kampouraki, A., Manis, G., & Nikou, C. (2008). Heartbeat time series classification with support vector machines. *IEEE transactions on information technology in biomedicine : a publication of the*

IEEE Engineering in Medicine and Biology Society, *13*, 512–8. https://doi.org/10.1109/TITB.2008.2003323

Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). *Visualizing and understanding recurrent networks*. arXiv: 1506.02078 [cs.LG].

Keogh, E. (2002). Exact indexing of dynamic time warping. *Proceedings of the 28th International Conference on Very Large Data Bases*, 406–417.

Kim, S. W., Zhou, Y., Philion, J., Torralba, A., & Fidler, S. (2020). *Learning to simulate dynamic environments with gamegan*. arXiv: 2005.12126 [cs.CV].

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling* (1st ed.). Springer, New York, NY. https://doi.org/10.1007/978-1-4614-6849-3

LeCun, Y. A., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade: Second edition* (pp. 9–48). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_3

LeCun, Y. A., & Cortes, C. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. http://yann.lecun.com/exdb/mnist/

Levison, W. H., Baron, S., & Kleinman, D. L. (1969). A model for human controller remnant. *IEEE Transactions on Man-Machine Systems*, *10*(4), 101–108. https://doi.org/10.1109/TMMS.1969.299906

Levison, W. H., Lancraft, R. E., & Junker, A. M. (1979). Effects of simulator delays on performance and learning in a roll-axis tracking task. *Proceedings of the 15th Annual Conference on Manual Control, Wright State Univ., Dayton, OH*, 168–186.

Li, C., Khan, L., & Prabhakaran, B. (2006). Real-time classification of variable length multi-attribute motions. *Knowl. Inf. Syst.*, *10*(2), 163–183. https://doi.org/10.1007/s10115-005-0223-8

Lim, B., Son, S., Kim, H., Nah, S., & Lee, K. M. Enhanced deep residual networks for single image super-resolution. In: *2017-July*. 2017, 1132–1140. https://doi.org/10.1109/CVPRW.2017.151.

Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing sax: A novel symbolic representation of time series. *Data Min. Knowl. Discov.*, *15*, 107–144. https://doi.org/10.1007/s10618-007-0064-z

Lin, M., Chen, Q., & Yan, S. (2014). *Network in network*. arXiv: 1312.4400 [cs.NE].

Lines, J., Taylor, S., & Bagnall, A. (2018). Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Trans. Knowl. Discov. Data*, *12*(5). https://doi.org/10.1145/3182382

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A. W. M., van Ginneken, B., & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, *42*, 60–88. https://doi.org/10.1016/j.media.2017.07.005

Lone, M., & Cooke, A. (2014). Review of pilot models used in aircraft flight dynamics. *Aerospace Science and Technology*, *34*, 55–74. https://doi.org/10.1016/j.ast.2014.02.003

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965

Lu, T., Pool, D. M., van Paassen, M. M., & Mulder, M. (2015). Use of simulator motion feedback for different classes of vehicle dynamics in manual control tasks. *Proceedings of the 5th CEAS Air & Space Conference, Delft, The Netherlands*.

Lundberg, S., & Lee, S.-I. (2017). A unified approach to interpreting model predictions.

Marina Martinez, C., Heucke, M., Wang, F., Gao, B., & Cao, D. (2018). Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey. *IEEE Transactions on Intelligent Transportation Systems*, *19*(3), 666–676. https://doi.org/10.1109/TITS.2017.2706978

Masters, D., & Luschi, C. (2018). *Revisiting small batch training for deep neural networks*. arXiv: 1804.07612 [cs.LG].

Mcculloch, W., & Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 127–147.

McRuer, D. T., Graham, D., Krendel, E. S., & W., R. J. (1965). *Human pilot dynamics in compensatory systems: Theory, models, and experiments with controlled element and forcing function variations*. Air Force Flight Dynamics Laboratory, Wrigth-Patterson AFB (OH).

McRuer, D. T., & Jex, H. R. (1967). A review of quasi-linear pilot models. *IEEE Transactions on Human Factors in Electronics*, *HFE-8, No. 3*, 231–249.

McRuer, D. T., & Krendel, E. S. (1959). The human operator as a servo system element. *J. Franklin Inst.*, *267*, pp. 381–403 May, pp. 511–536 June.

McRuer, D. T., Weir, D. H., & Klein, R. H. (1971). A pilot-vehicle systems approach to longitudinal flight director design. *Journal of Aircraft*, *8*(11), 890–897. https://doi.org/10.2514/3.59186

Minsky, M., & Papert, S. A. (1969). *Perceptrons*. MIT Press.

Mittelman, R. (2015). *Time-series modeling with undecimated fully convolutional neural networks*. arXiv: 1508.00317 [stat.ML].

Mohan, A. T., & Gaitonde, D. V. (2018). *A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks*. arXiv: 1804.09269 [physics.comp-ph].

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines, 807–814.

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press.

Nieuwenhuizen, F. M., Mulder, M., van Paassen, M. M., & Bülthoff, H. H. (2013). Influences of simulator motion system characteristics on pilot control behavior. *Journal of Guidance, Control, and Dynamics*, *36*(3), 667–676. https://doi.org/10.2514/1.59257

Nittala, S. K. R., Elkin, C. P., Kiker, J. M., Meyer, R., Curro, J., Reiter, A. K., Xu, K. S., & Devabhaktuni, V. K. (2018). Pilot skill level and workload prediction for sliding-scale autonomy. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 1166–1173. https://doi.org/10.1109/ICMLA.2018.00188

Ordóñez, F. J., & Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, *16*(1). https://doi.org/10.3390/s16010115

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks.

Patterson, J., & Gibson, A. (2017). *Deep learning: A practioner's approach* (First). O'Reilly Media, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pool, D. M., Harder, G. A., & van Paassen, M. M. (2016). Effects of simulator motion feedback on training of skill-based control behavior. *Journal of Guidance, Control, and Dynamics*, *39*(4), 889–902. https://doi.org/10.2514/1.G001603

Pool, D. M., & Zaal, P. M. T. (2016). A cybernetic approach to assess the training of manual control skills. *IFAC-PapersOnLine*, *49-19*, 343–348.

Pool, D. M., Zaal, P. M. T., van Paassen, M. M., & Mulder, M. (2010). Effects of heave washout settings in aircraft pitch disturbance rejection. *Journal of Guidance, Control, and Dynamics*, *33*(1), 29–41. https://doi.org/10.2514/1.46351

Pool, D. M., Pais, A., Vroome, A., van Paassen, M. M., & Mulder, M. (2012). Identification of nonlinear motion perception dynamics using time-domain pilot modeling. *Journal of Guidance, Control, and Dynamics*, *35*, 749–763. https://doi.org/10.2514/1.56236

Pool, D. M., Zaal, P. M. T., Damveld, H., van Paassen, M. M., Vaart, J., & Mulder, M. (2011). Modeling wide-frequency-range pilot equalization for control of aircraft pitch dynamics. *Journal of Guidance, Control, and Dynamics*, *34*(5), 1529–1542. https://doi.org/10.2514/1.53315

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions.

Ramakrishnan, N., & Soni, T. (2018). Network traffic prediction using recurrent neural networks. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 187–193. https://doi.org/10.1109/ICMLA.2018.00035

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. A. (2007). Efficient learning of sparse representations with an energy-based model. *Advances in Neural Information Processing Systems 19*, 1137–1144.

Rasmussen, J. (1983). Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*(3), 257–266. https://doi.org/10.1109/TSMC.1983.6313160

Ratanamahatana, C., & Keogh, E. (2005). Three myths about dynamic time warping data mining. *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, 506–510. https://doi.org/10.1137/1.9781611972757.50

Reimers, N., & Gurevych, I. (2017). *Optimal hyperparameters for deep lstm-networks for sequence labeling tasks*. arXiv: 1707.06799 [cs.CL].

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144. https://doi.org/10.1145/2939672.2939778

Rolnick, D., Veit, A., Belongie, S., & Shavit, N. (2018). Deep learning is robust to massive label noise.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. Mcclelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations* (pp. 318–362). MIT Press.

Saito, S., Simon, T., Saragih, J., & Joo, H. (2020). Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Saleh, K., Hossny, M., & Nahavandi, S. (2017). Driving behavior classification based on sensor data fusion using lstm recurrent neural networks. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 1–6. https://doi.org/10.1109/ITSC.2017.8317835

Shorten, C., & Khoshgoftaar, T. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, *6*. https://doi.org/10.1186/s40537-019-0197-0

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359. https://doi.org/10.1038/nature24270

Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*. arXiv: 1409.1556 [cs.CV].

Smith, L. N. (2017). *Cyclical learning rates for training neural networks*. arXiv: 1506.01186 [cs.CV].

Socher, R., Lin, C. C.-Y., Ng, A. Y., & Manning, C. D. (2011). Parsing natural scenes and natural language with recursive neural networks. *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 129–136.

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*(3), 714–735. https://doi.org/10.1109/72.572108

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Stapleford, R., Peters, R., & Alex, F. (1969). Experiments and a model for pilot dynamics with visual and motion inputs. nasa cr-1325. *NASA contractor report. NASA CR. United States. National Aeronautics and Space Administration*, 1–115.

Steurs, M., Mulder, M., & van Paassen, M. M. ((2004). A cybernetic approach to assess flight simulator fidelity. *AIAA Modeling and Simulation Technologies Conference and Exhibit*. https://doi.org/10.2514/6.2004-5442

Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). *A survey of optimization methods from a machine learning perspective*. arXiv: 1906.06821 [cs.LG].

Sundararajan, M., Taly, A., & Yan, Q. (2017). *Axiomatic attribution for deep networks*. arXiv: 1703.01365 [cs.LG].

Susto, G. A., Cenedese, A., & Terzi, M. (2018). Time-series classification methods: Review and applications to power systems data. https://doi.org/10.1016/B978-0-12-811968-6.00009-7

Sutskever, I. (2013). *Training recurrent neural networks* (Ph.D. dissertation). University of Toronto.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Tango, F., & Botta, M. (2013). Real-time detection system of driver distraction using machine learning. *IEEE Transactions on Intelligent Transportation Systems*, *14*(2), 894–905. https://doi.org/10.1109/TITS.2013.2247760

Tieleman, T., & Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.

Um, T. T., Pfister, F. M. J., Pichler, D., Endo, S., Lang, M., Hirche, S., Fietzek, U., & Kulić, D. (2017). Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, 216–220. https://doi.org/10.1145/3136755.3136817

Umpleby, S. A. (2005). A history of the cybernetics movement in the united states. *Journal of the Washington Academy of Sciences*, *91, No. 2*, 54–66.

van der El, K., Pool, D. M., & Mulder, M. (2019). Analysis of human remnant in pursuit and preview tracking tasks [14th IFAC Symposium on Analysis, Design, and Evaluation of Human Machine Systems HMS 2019]. *IFAC-PapersOnLine*, *52*(19), 145–150. https://doi.org/https://doi.org/10.1016/j.ifacol.2019.12.165

van der Vaart, J. C. (1992). *Modelling of perception and action in compensatory manual control tasks* (Ph.D. thesis). Delft University of Technology, Faculty of Aerospace Engineering. http://repository.tudelft.nl

Versteeg, R. (2019). *Classifying human control behavior by artificial intelligence* (MSc. thesis). Delft University of Technology, Faculty of Aerospace Engineering. http://repository.tudelft.nl

Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L. (2019). Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, *119*, 3–11. https://doi.org/10.1016/j.patrec.2018.02.010

Wang, W., & Gang, J. (2018). Application of convolutional neural network in natural language processing. *2018 International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 64–70. https://doi.org/10.1109/ICISCAE.2018.8666928

Wang, X., & Gupta, A. (2015). Unsupervised learning of visual representations using videos.

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. *2017 International Joint Conference on Neural Networks (IJCNN)*, 1578–1585. https://doi.org/10.1109/IJCNN.2017.7966039

Wen, Q., Sun, L., Yang, F., Song, X., Gao, J., Wang, X., & Xu, H. (2021). *Time series data augmentation for deep learning: A survey*. arXiv: 2002.12478 [cs.LG].

Wiener, N. (1948). *Cybernetics: Or control and communication in the animal and the machine* (2nd ed.). MIT Press.

Wijlens, R., Zaal, P. M. T., & Pool, D. M. (2020). Retention of manual control skills in multi-axis tracking tasks. *Aiaa scitech 2020 forum* (pp. 1–26). American Institute of Aeronautics; Astronautics Inc. (AIAA). https://doi.org/10.2514/6.2020-2264

Wu, R., Yan, S., Shan, Y., Dang, Q., & Sun, G. (2015). *Deep image: Scaling up image recognition*. arXiv: 1501.02876 [cs.CV].

Xi, P., Law, A., Goubran, R., & Shu, C. (2019). Pilot workload prediction from ecg using deep convolutional neural networks. *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, 1–6. https://doi.org/10.1109/MeMeA.2019.8802158

Xu, S., Tan, W., Efremov, A., Sun, L., & Qu, X. (2017). Review of control models for human pilot behavior. *Annual Reviews in Control*, *44*, 274–291. https://doi.org/10.1016/j.arcontrol.2017.09.009

Yao, H., Zhang, X., Zhou, X., & Liu, S. (2019). Parallel structure deep neural network using cnn and rnn with an attention mechanism for breast cancer histology image classification. *Cancers*, *11*(12). https://doi.org/10.3390/cancers11121901

Yoon, J., Jarrett, D., & van der Schaar, M. (2019). Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, *32*, 5508–5518.

Zaal, P. M. T., Pool, D. M., de Bruin, J., Mulder, M., & van Paassen, M. M. (2009). Use of pitch and heave motion cues in a pitch control task. *Journal of Guidance, Control, and Dynamics*, *32*(2), 366–377. https://doi.org/10.2514/1.39953

Zeiler, M. D. (2012). *Adadelta: An adaptive learning rate method*. arXiv: 1212.5701 [cs.LG].

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). Time series classification using multi-channels deep convolutional neural networks. *WAIM 2014. LNCS*, *8485*, 298–310.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2921–2929. https://doi.org/10.1109/CVPR.2016.319

Zhu, J., Chen, N., & Peng, W. (2019). Estimation of bearing remaining useful life based on multiscale convolutional neural network. *IEEE Transactions on Industrial Electronics*, *66*(4), 3208–3216. https://doi.org/10.1109/TIE.2018.2844856

Zollner, H. G. H., Pool, D. M., Damveld, H. J., van Paassen, M. M., & Mulder, M. (2010). The effects of controlled element break frequency on pilot dynamics during compensatory target-following. *AIAA Modeling and Simulation Technologies Conference*, 1–12. https://doi.org/10.2514/6.2010-8092