Machine Learning (2021) 110:3037–3057
https://doi.org/10.1007/s10994-021-06055-x

Deep learning and multivariate time series for cheat detection in video games

José Pedro Pinto[2] · André Pimenta[1] · Paulo Novais[2]

Received: 1 March 2021 / Revised: 1 July 2021 / Accepted: 9 August 2021 /
Published online: 14 October 2021

Abstract
Online video games drive a multi-billion dollar industry dedicated to maintaining a competitive and enjoyable experience for players. Traditional cheat detection systems struggle when facing new exploits or sophisticated fraudsters. More advanced solutions based on machine learning are more adaptive but rely heavily on in-game data, which means that each game has to develop its own cheat detection system. In this work, we propose a novel approach to cheat detection that doesn't require in-game data. Firstly, we treat the multimodal interactions between the player and the platform as multivariate time series. We then use convolutional neural networks to classify these time series as corresponding to legitimate or fraudulent gameplay. Our models achieve an average accuracy of respectively 99.2% and 98.9% in triggerbot and aimbot (two widespread cheats), in an experiment to validate the system's ability to detect cheating in players never seen before. Because this approach is based solely on player behavior, it can be applied to any game or input method, and even various tasks related to modeling human activity.

Keywords Deep learning · Multivariate time series · Human–computer interaction · Video games

Editors: João Gama, Alípio Jorge, Salvador García.

José Pedro Pinto
a80741@alunos.uminho.pt

André Pimenta
apimenta@anybrain.gg

Paulo Novais
pjon@di.uminho.pt

1 Anybrain, S.A., Braga, Portugal

2 Universidade do Minho, Braga, Portugal

1 Introduction

Video games have been rapidly growing in popularity in the past two decades, driving what is nowadays a multi-billion dollar industry. Given how widespread this form of entertainment is, we ought to consider the social implications of online video-games, especially in situations such as the current global pandemic, when this form of entertainment is also one of the remaining means of social connection.

Developers and publishers of online video games focus on maintaining healthy communities. One aspect that contributes to the engagement of players is the competitiveness of the game. Cheaters are players who resort to exploits or third-party software to gain unfair advantages, often disturbing the game's competitiveness.

In this sense, great effort has gone into developing anti-cheating systems. These systems assert if a player cheated during a given match or time period.

Anti-cheating systems help to provide a better experience to the players. By keeping the players engaged in a competitive environment, the game communities can keep growing, providing entertainment, and generating profit.

The problem with traditional anti-cheating systems is that they have a history of always being one step behind the most sophisticated fraudsters and cheaters. Most of them consist of searching for malware or evidence that the game software has been tampered with.

In this sense, machine learning has helped by providing a statistical approach and tools to predict if a player is cheating based on his data (Yeung et al., 2006; Galli et al., 2011; Alkhalifa, 2016; Islam et al., 2020; Alayed et al., 2013).

Nearly all machine learning approaches to anti-cheating in video games consist of analyzing in-game data, which is information regarding the game environment (such as the player's avatar positioning or activity during gameplay). Analyzing this data demands domain knowledge and a process of feature engineering for each game.

A system capable of analyzing gameplay without relying on in-game data would hold great value since it could be applied to several games without modification and be adaptive to new types of cheats. We developed an anti-cheating system that analyzes the interaction between the player and the computer, attempting to learn which behavioral patterns occur when a player is cheating.

Human–computer interaction (HCI) data can be very complex, and one possible approach is to compute an array of behavioral biometrics for fixed time windows (Pimenta et al., 2015, 2014; Carneiro et al., 2016). In this work, we take a different approach and build multivariate time series to construct a more detailed representation of the interactions.

Multivariate time series are difficult to analyze, especially when they do not contain regular and periodic phenomena. Human behavior can be highly irregular and spontaneous, which makes this task a challenge. To recognize patterns in this data, we use convolutional neural networks, which have been successful in many problems involving sequential data.

Our core hypothesis is that cheaters share behavioral patterns that we can represent with multivariate time series and successfully recognize with deep learning models.

To the best of our knowledge, this is the first work that approaches cheat detection from an HCI perspective and successfully employs deep learning and multivariate time series in cheat detection for video games. We tested our approach in two widespread cheats for the famous game Counter-Strike: Global Offensive, with a dataset containing 490 h of real human interactions from 118 players.

The presented framework portrays an outstanding performance in cheat detection, but most importantly it accomplishes that by relying on HCI events (such as keystrokes and mouse clicks) that are ubiquitous not just in video games, but in the way humans interact with computers.

Our contributions can be summarized as follows:

- Novel approach to cheat detection in video games that relies only on input data such as keystrokes and mouse movements;
- CNN architecture to detect cheating occurrences in a supervised fashion;
- Data collection and processing method that creates a multivariate time series representation of HCI.

As for the organization of this paper, we start by introducing some concepts essential to our work. We then present related scientific work regarding cheat detection in video games. Next, we present our novel approach, namely our data collection methods and the adopted deep learning architecture. Afterward, we explain the conducted experiments and report the obtained results. Finally, we discuss those results and make some final remarks regarding this work's conclusion and future work.

2 Fundamental concepts

In this section, we introduce some concepts that are essential to the approach we propose.

2.1 Multivariate time series

We define a time series $\mathbf{X}$ in Eq. 2 as a sequence of T observations $\mathbf{x}_t$ of a given state at time t, called timesteps. Each timestep $\mathbf{x}_t$ is composed of D values, as shown in Eq. 1. If $D > 1$, we face a multivariate timeseries, represented in Equation.

$$\mathbf{x}_t = \{x_{t1}, x_{t2}, x_{t3}, ..., x_{tD}\} \tag{1}$$

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_T\} \tag{2}$$

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1D} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2D} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3D} \\ \dots & \dots & \dots & \dots & \dots \\ x_{T1} & x_{T2} & x_{T3} & \dots & x_{TD} \end{bmatrix} \tag{3}$$

This data structure can represent dynamic systems across many fields. In an industrial context, the multivariate time series can store data collected from an ensemble of sensors monitoring a production pipeline (Siegel, 2020; Liu et al., 2019; Mehdiyev et al., 2017; Filonov et al., 2016). In finance, each dimension can contain a stock's price or trade volume fluctuation (Mehtab & Sen, 2020). Health-care is another important use-case, where each timestep can contain measurements regarding biometrics such as heart rate and blood pressure or gait-related features (Tan et al. 2019).

As pointed out by Xing et al. (2010), we can divide sequential data analysis methods into three categories: feature-based, distance-based, and model-based.

The first approach consists of processing sequences into a feature vector that can be analyzed by classifiers such as decision trees or SVMs. These methods are prone to lose information and generally do not preserve the ordered nature of sequences.

Distance-based methods rely on distance functions such as Dynamic Time Warping (DTW) or the Wasserstein distance. These can serve as the underlying similarity measures in clustering methods such as K-Means Clustering or K-Nearest Neighbors.

Model-based approaches attempt to build a parametric approximation of the probability density function of the sequences. Traditionally, models such as Dynamic Bayesian Networks (DBNs) or Hidden Markov Models (HMMs) were popular due to their simplicity. More recently, with the advent of Deep Learning, models such as convolutional neural networks (CNN) and Long Short-Term Memory (LSTM) recurrent neural networks have increasingly become more popular for sequential data-related tasks.

In Table 1 we can find previous work in multivariate time series analysis, grouped by task.

In our study of previous work involving multivariate time series, we find that LSTM architectures perform well when the data portrays some sort of perioding and regular variation. Pattern recognition in more erratic time series is usually more successful with CNN architectures.

2.2 Convolutional neural networks

CNNs (Cun et al., 1990) have been successful in many tasks, being particularly popular for their use in computer vision. These models employ an infinitely strong prior belief that features interact only within a given range of proximity (to which we refer in practice as

Table 1 Previous related work involving multivariate time series

Task	Paradigm	Work	Models used
Anomaly detection in multivariate time series	Unsupervised	Siegel (2020)	RNN, CNN
		Lu et al. (2017)	RNN
		Lin et al. (2020)	LSTM
		Dietterich (2002)	CNN
Multivariate time series classification	Supervised	Zheng et al. (2016)	CNN
		Cui et al. (2016)	RNN
		Liu et al. (2019)	CNN
		Zhao et al. (2017)	CNN
		Wang et al. (2017)	CNN
		Karim et al. (2018)	LSTM + CNN
		Tan et al. (2019)	LSTM
Multivariate time series forecasting	Supervised	Borovykh et al. (2018)	CNN
		Wan et al. (2019)	CNN
		Mehtab and Sen (2020)	CNN
		Filonov et al. (2016)	LSTM
		Du et al. (2018)	LSTM
		Sagheer and Kotb (2019)	LSTM

convolutional layers' kernel size). As pointed out by Goodfellow et al. (2016), CNNs portrays three essential properties:

- Firstly, they build an equivariant representation, since they apply the same kernel to all input locations, which means that the interaction between features occurs in the same way regardless of location;
- The parameters that encode equivariant interactions are shared across all input locations. Parameter sharing acts as a regulatory factor;
- Sparse connectivity greatly reduces the number of parameters in the model, which facilitates its training.

These properties make CNNs a great candidate model for our approach since we intend to recognize cheating patterns regardless of when they occur in the interaction. Additionally, the statistical efficiency of CNNs is of great value in data as complex as multivariate time series and as irregular as human behavior can be.

Methods used in previous work do not take advantage of the temporal context in time series, so they would not be of use when we formulate our problem with this data structure. An alternative to CNNs would be recurrent neural networks, especially gated architectures which are feature in work referenced in Table 1. RNNs, however, are exhibit some disadvantages that make them an inferior choice. Due to unfolding, RNNs tend to have an extremely high number of parameters and are difficult to paralelize (time steps need to be processed sequentially).

2.3 Deep learning regularization techniques

There are many methods applied to deep learning models in order to lower their generalization error without harming their ability to learn complex functions. Here we review the main techniques which we used in this work.

L2 Regularization

Parameter norm penalties consist of introducing a penalty in the objective function of the algorithm with the purpose of stabilizing the model's parameters' values. If our model is described by the set of parameters $\boldsymbol{\theta}$, and we have an objective function $J(\boldsymbol{\theta};\boldsymbol{X})$, the regularized function $\tilde{J}$ can be defined as

$$\tilde{J}(\boldsymbol{\theta};\boldsymbol{X}) = J(\boldsymbol{\theta};\boldsymbol{X}) + \alpha\Omega(\boldsymbol{\theta}) \tag{4}$$

where Ω is the function we use to calculate parameter norm and α is a hyperparameter that defines how strong we want the penalty to be. The L2 regularization is the most popular parameter norm penalty, defined by

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{w_\theta}||_2^2 \tag{5}$$

$$\tilde{J}(\boldsymbol{\theta};\boldsymbol{X}) = J(\boldsymbol{\theta};\boldsymbol{X}) + \lambda||\boldsymbol{w_\theta}||_2^2 \tag{6}$$

where $\boldsymbol{w_\theta}$ is the set of trainable parameters in $\boldsymbol{\theta}$. Notice that in Eq. 6, the value of λ replaces what would be $\frac{\alpha}{2}$ according to Eqs. 5 and 4, for a matter of simplicity.

Dropout

Given the stochastic factors in the training of deep learning models (in parameter initialization or data sampling, for example), it is extremely unlikely that an algorithm converges to the same model twice.

In this sense, we can train several models using the same exact process. If these models make independent errors, we can combine their answers to make even better predictions.

This is the core idea in bagging (Breiman, 1996), a method that aggregates the answers of several models under the assumption that they make independent errors. These models are trained with datasets randomly selected with replacement from the original dataset. If the errors are correlated, bagging performs on average at least as well as any of the models in the ensemble.

Another regularization method that can be seen as bagging is called dropout (Srivastava et al., 2014), which works by randomly dropping connections in a neural network at train time, as illustrated in Fig. 1. Dropout provides a way to train many less dense models without the significant computational overhead associated with other bagging methods.

The ability to train these models simultaneously and implicitly is due to the parameter sharing between them. This parameter sharing motivates the whole network to learn redundant units since if a certain feature is not present, the network can still make a correct prediction based on other information.

3 Related work

Machine learning has been used for cheat detection in video games before. Table 2 summarizes previous work from which we can draw a few conclusions.

Nearly all approaches rely on in-game data (information regarding the state of the player's avatar in the game environment) and follow the supervised learning paradigm. Additionally, the only works using time series do not explore deep learning models, which are currently the most popular methods for dealing with multivariate time series. From the

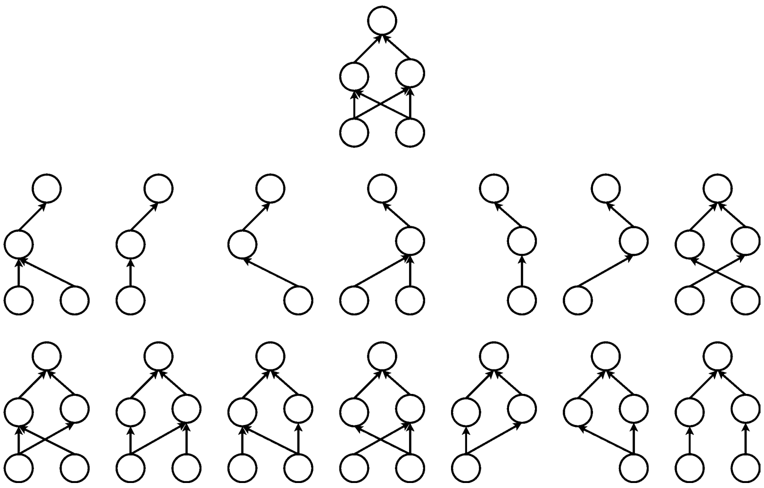

Fig. 1 Possible architectures generated by the dropout regularization technique in a very simple neural network

Table 2 Related work applying machine learning to cheat detection in video games

Data origin	Data structure	Paradigm	Work	Models used
Network traffic data	Tabular data	Unsupervised	Islam et al. (2020)	Kernel machines variant
In-game data	Tabular data	Supervised	Galli et al. (2011)	SVM logistic regression
			Alayed et al. (2013)	Decision trees random forest MLP SVM
			Pao et al. (2010)	SVM K-nearest neighbors
In-game data	Multivariate time series	Supervised	Yeung et al. (2006)	DBN
			Alkhalifa (2016)	HMM

analyzed related work, Islam et al. (2020) stands out as the only unsupervised learning approach and the only one that doesn't use in-game data. Their approach consists of trying to detect patterns in network traffic that correlate with cheating.

In this sense, there are two aspects in our approach that we couldn't find in any previous scientific work or report:

- Use of deep learning and multivariate time series for cheat detection in video games;
- Approaching cheat detection from an HCI perspective and capturing the interaction between the player and the platform.

4 The proposed approach

In this section, we describe our novel approach to cheat detection, based on the idea of representing HCI with multivariate time series. First, we provide details on the data collection and preprocessing methods. Then, we specify the architecture used for classification.

4.1 Data collection

Our cheat detection system relies on a generic approach to player behavior. Instead of analyzing contextual game data (such as avatar positioning and activity), we analyze the events produced by the platform's peripherals (keystrokes and mouse activity, for example). As we've mentioned, this allows the application of this system to several contexts (different games or even domains beyond gaming).

This data collection method greatly diverges from previous work in cheat detection using machine learning, since it does not rely on the game engine to retrieve any sort of contextual data.

Carneiro et al. (2016), Pimenta et al. (2015, 2014), we see a similar approach to assess mental fatigue in computer users. In these works, hardware events were processed to produce a collection of behavioral biometrics. These biometrics were calculated based on domain knowledge and consisted of aggregational descriptive statistics.

Table 3 Events occurring during an interaction

Timestamp (ms)	Code	Value	Event description
1574365839854	cursorX	960	The cursor's horizontal location
1574365839854	D	1	The D keyboard key was pressed
1574365839854	cursorY	540	The cursor's vertical location
1574365839854	MOUSE_LEFT	1	The left mouse button was pressed
1574365840188	A	1	The A keyboard key was pressed
1574365840196	D	0	The D keyboard key was released
1574365840392	cursorX	960	The cursor's horizontal location
1574365840392	cursorY	540	The cursor's vertical location
1574365840392	MOUSE_LEFT	0	The left mouse button was released
…	…	…	…

This table contains roughly 500 ms of a real interaction from our data

Table 4 Multivariate time series of the events in Table 3

Timestamp	A	D	MOUSE_LEFT	cursorX_var	cursorY_var
1574365839854	0	1	1	0	0
1574365839954	0	1	1	0	0
1574365840054	0	1	1	0	0
1574365840154	0.66	0.42	1	0	0
1574365840254	1	0	1	0	0
1574365840354	1	0	0.38	0	0
…	…	…	…	…	…

In this work, we make use of raw event data with minor preprocessing. Table 3 illustrates the information we collect for each event. Each event is characterized by three attributes: a timestamp, a code, and a value.

Once we have the collection of events described above, we process them to obtain a data structure as seen in the sample in Table 4. Each column represents an event code and each row represents a timestep (each timestep corresponds to 100 ms).

We aggregate events in timesteps according to the value range of each event code.

For binary events (such as pressing or releasing keys), the value for each timestep is the activation time (e.g. duration of a key press) within that timestep, given by the formula

$$x_t = \frac{\sum_n \left[\min(t+T,\ e_{up_n}) - \max(t,\ e_{down_n})\right]}{T}, \qquad \forall n\ :\ t < e_{up_n} < t+T\ \vee\ t < e_{down_n} < t+T \tag{7}$$

where e_{up_n} and e_{down_n} are the timestamps between which the nth event with code e occured. T is the duration of a timestep and t is the beginning of the timestep.

For real-valued columns, we follow a different approach. For each possible event code, we generate two features expressing the variance and the amplitude of the values in each timestep. Following the same language as before, the two columns can be defined with the formulae

$$\begin{cases} x_{t_{var}} = \sigma\left(\frac{\sum_{n=1}^{N}(e_{n_{value}} - \overline{e_{values}})^2}{N-1}\right) \\ x_{t_{amp}} = \sigma\left(\max_n e_{n_{value}} - \min_n e_{n_{value}}\right) \end{cases}, \qquad (8)$$
$$\forall\, n \;:\; t < e_n < t + T$$

where $e_{n_{value}}$ is the value of the n^{th} event with the keycode of the column being calculated. The function σ serves the purpose of keeping the values of these columns in the same range as the binary columns, and is defined by the formula

$$\sigma : \quad [-\infty,\ +\infty] \ \rightarrow \ [-1,\ 1]$$
$$\sigma(x) = \frac{2}{1 + \exp(-x)} - 1 \qquad (9)$$

We can visualize the transformation applied by σ in Figs. 2 (before) and 3 (after). The mouse movement features we used in this work assume values in $[0,\ +\infty[$. When we apply the σ function, these features are squashed into the interval $[0,\ 1[$, which is nearly same domain seen in the features resulting from binary events.

4.2 Classifier architecture

Our proposed architecture for classification is a feedforward convolutional neural network illustrated in Fig. 4. We follow each layer except for the last with a dropout mask to achieve

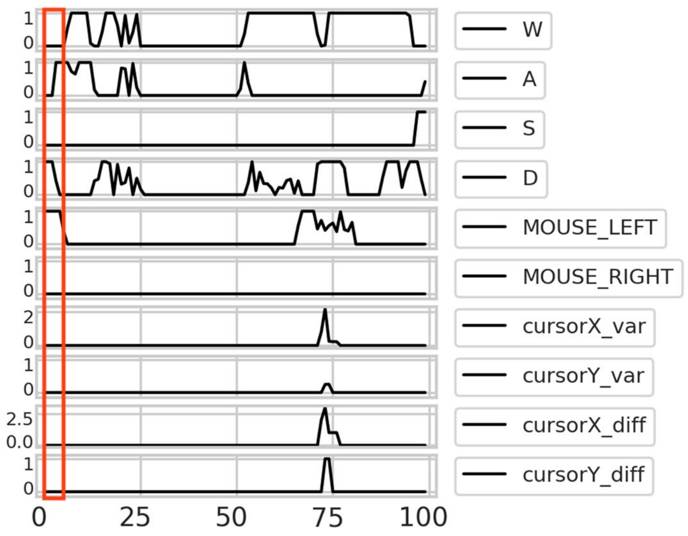

Fig. 2 Example of 10 s of interaction between the player and the computer. The red rectangle highlights the data sample seen in Tables 3 and 4

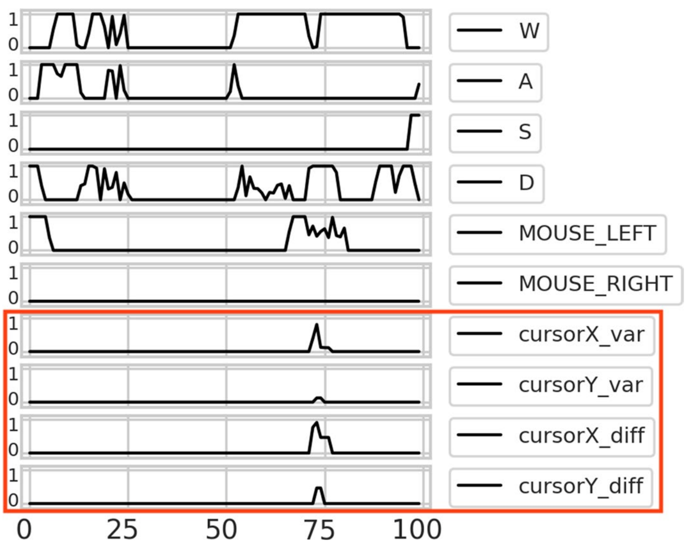

Fig. 3 The same data seen in Fig. 2, after applying the function σ to the features outlined by the red rectangle. Notice that the values of these features now lie between 0 and 1

better generalization. The use of a small number of filters can also be seen as a form of obtaining better generalization.

Filter size has a strong meaning in this context because the range of timesteps interacting to originate a hidden feature determines the time period that influences that feature.

We can multiply that range by applying some sort of pooling operation, such as max-pooling, and although this can also mean fewer parameters and a model that is easier to train, models using max-pooling layers did not perform as well as the ones that don't use pooling.

Perhaps in tasks with a necessity for detecting patterns with a wide temporal range, deeper models that make use of pooling might be the better choice.

5 Experiments

In this section, we explain our experiments. First, we describe our dataset to show that our results don't just apply to a controlled environment as seen in previous work but extend to a real-world scenario. We then present the results that led to our proposed architecture and finally present a method for player-based cross-validation that allowed us to assert if our approach can deal with unseen players.

5.1 Dataset description

The dataset we used in our experiments was collected in a real-world context of players in the game Counter-Strike: Global Offensive, a first-person shooter. Players installed an application that collected keyboard and mouse events as previously described. Since we

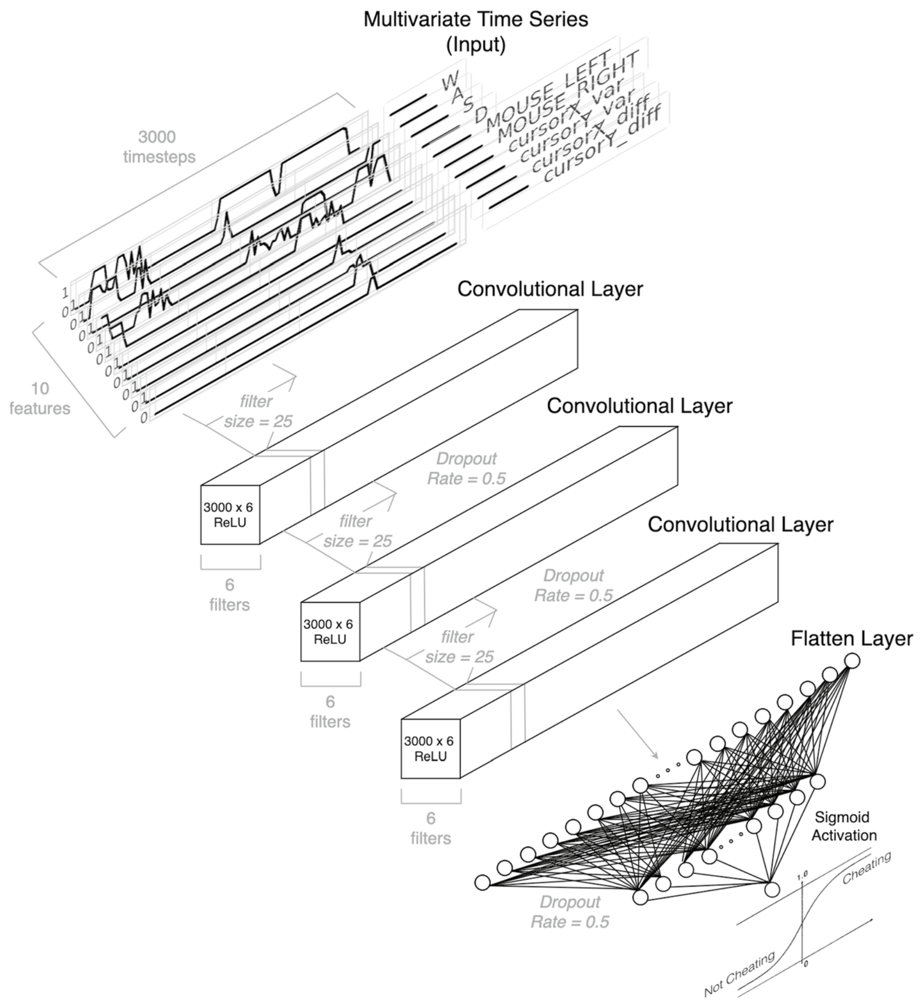

Fig. 4 Proposed architecture for the CNN classifier

intended our dataset to be as realistic as possible, most data resulted from normal players (not cheaters) engaging in matches on the game's official servers. Players were given full freedom to play as they intended (as long as they were not cheating) to maximize behavior variety in the dataset.

We tested two types of cheating:

- Aimbot—a cheat that automatically aims towards the cheater's target, thus greatly reducing the need to perform mouse movements;
- Triggerbot—a script that automatically fires the weapon as soon as the crosshair reaches an opponent, thus reducing the need for a fast reaction.

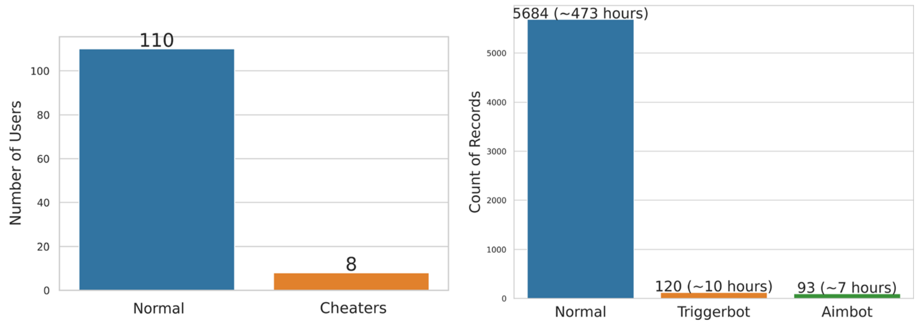

Fig. 5 Distribution of players and data records by cheat. Although there isn't an official report of the exact amount of active cheating players in this game, we can argue that this our dataset is representative of the real world proportion of legitimate to fraudulent players. Although cheaters form a minority of the community, it is fairly common for most players to encounter one or more cheaters in a match

These cheats greatly alter player behavior and allow much better performance in the game. The hypothesis motivating our approach is that these different behavioral patterns reflect in the multivariate time series, and are detected by the deep learning models.

As shown in Fig. 5, our dataset contains interaction data from 118 players, 8 of whom have engaged in cheating. Labels of cheating interactions are rare, which is coherent with a real-world scenario where cheaters represent a small minority of players.

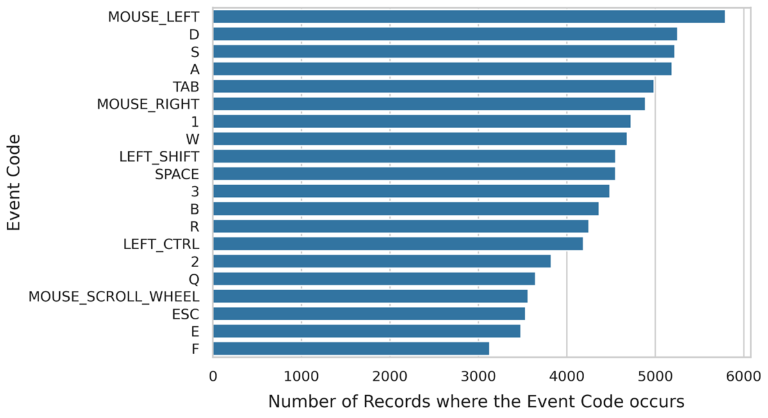

Fig. 6 Event codes that appear the most often in our dataset

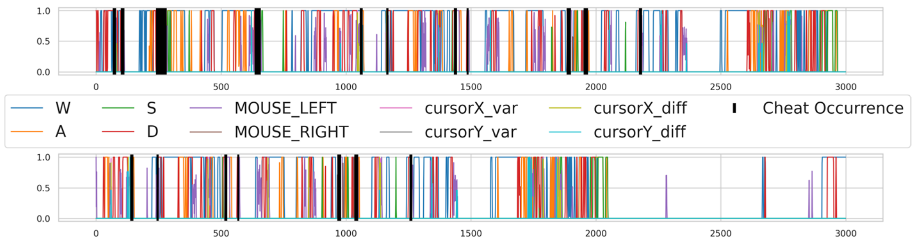

Fig. 7 Two examples of records in our dataset

The labels were generated in scheduled matches by altering the cheating software to produce a timestamp for every cheat activation. Each record in our dataset corresponds to 5 minutes of gameplay. We label a record as a cheating record if there is at least one activation of the cheat during that interaction. In Fig. 7 we can visualize two samples from our training dataset. Each record is a multivariate time series with 3000 time steps and 10 values in each timestep. We selected those 10 variables based on the frequence with which they occur, as seen in Fig. 6, and on their respective performed function in the game.

5.2 Model hyperparameters

To arrive at our proposed architecture, we conducted several trials to search for a combination of hyperparameters that maximized the area under the receiver operating characteristic curve (AUC) metric.

We used TensorFlow (Abadi et al., 2015) and Keras (Chollet et al., 2015) for model implementation and Optuna (Akiba et al., 2019) for hyperparameter optimization. We used the Adam algorithm (Kingma & Ba, 2014) to optimize our models by minimizing L2 regularized cross-entropy, defined by

$$L(\boldsymbol{\theta};\boldsymbol{X};\boldsymbol{y}) = \lambda\|\boldsymbol{w_\theta}\|_2^2 - \frac{\sum_{i=1}^{N}\left[\boldsymbol{y}_i \times \log(p(\boldsymbol{X}_i)) + (1-\boldsymbol{y}_i) \times \log(1-p(\boldsymbol{X}_i))\right]}{N} \tag{10}$$

where $\boldsymbol{y}_i$ and $p(\boldsymbol{x}_i)$ are the true and predicted labels of the sample $\boldsymbol{X}_i$, respectively, N is the number of samples in $\boldsymbol{X}$, λ is the L2 regularization hyperparameter and $\boldsymbol{w_\theta}$ is the trainable subset of the model's parameters $\boldsymbol{\theta}$.

Table 5 describes our hyperparameter search space. The number of layers and filters in each layer was intended to regulate complexity in our model. The filter size has an impact on model complexity, but it also determines the temporal range of the interactions between features. The batch size and the L2 λ value were the variable sources of regularization (we also used a fixed 0.5 dropout rate in every layer). We also tested the use of max-pooling to explore the potential benefits of dimensionality reduction in hidden features. In those models, we applied the max-pooling operation following each convolutional layer.

In Figs. 8 and 9, we can observe the results of the hyperparameter optimization. It appears that two separate clusters are being formed according to the usage of max-pooling. In Fig. 10, we examine the distribution of the AUC metric and the loss function values in

Table 5 Hyperparameter search space and chosen values

Hyperparameter	Type	Search interval or set	Chosen value
L2 regularization λ	Real-valued	$\left[5 \times 10^{-5},\ 5 \times 10^{-3}\right]$	3×10^{-3}
Number of convolutional layers	Integer	[2, 8]	3
Number of filters	Integer	[4, 16]	6
Filter size	Integer	[10, 50]	25
Max pooling	Boolean	$\{True,\ False\}$	*False*
Batch size	Integer	[32, 512]	64

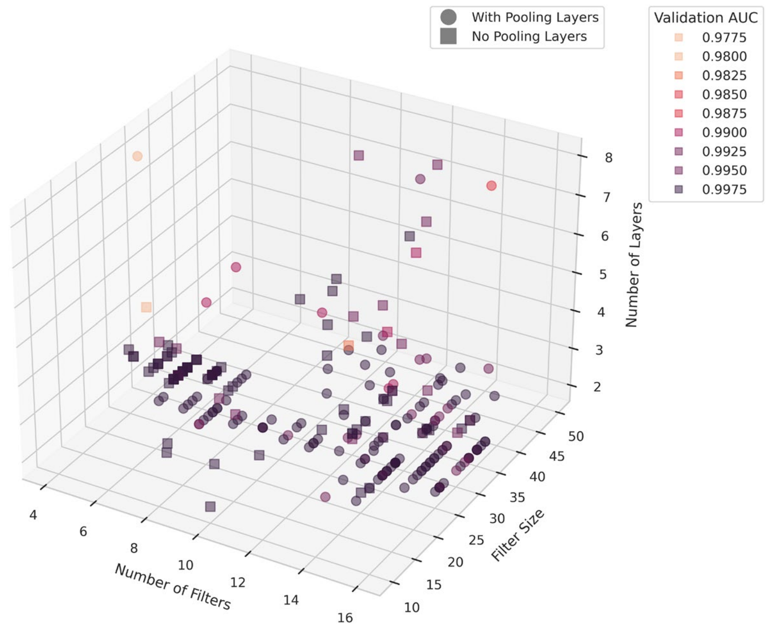

Fig. 8 Distribution of the experimental results by number of layers, number of filters in each layer, and filter size. The best trials correspond to CNNs with a filter size bewteen 20 and 30. In terms of number of layers and filters, we can observe two successful clusters of models: one with 2 layers and a higher number of filters (around 14 per layer), and other with 3 layers but a lower number of filters (around 6 per layer)

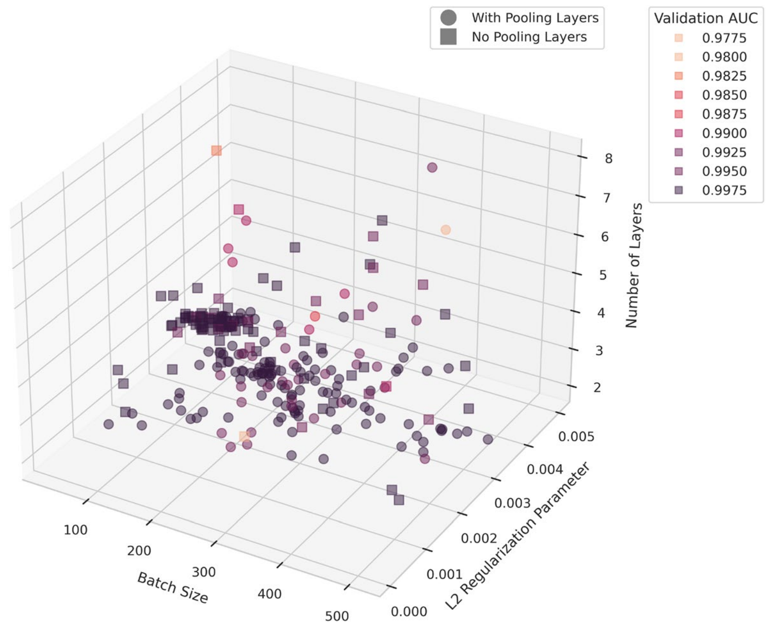

Fig. 9 Distribution of the experimental results by batch size, L2 Regularization parameter, and number of layers. The best trials featured a batch size between 100 and 200 and an L2 regularization parameter between 1e−3 and 2e−3

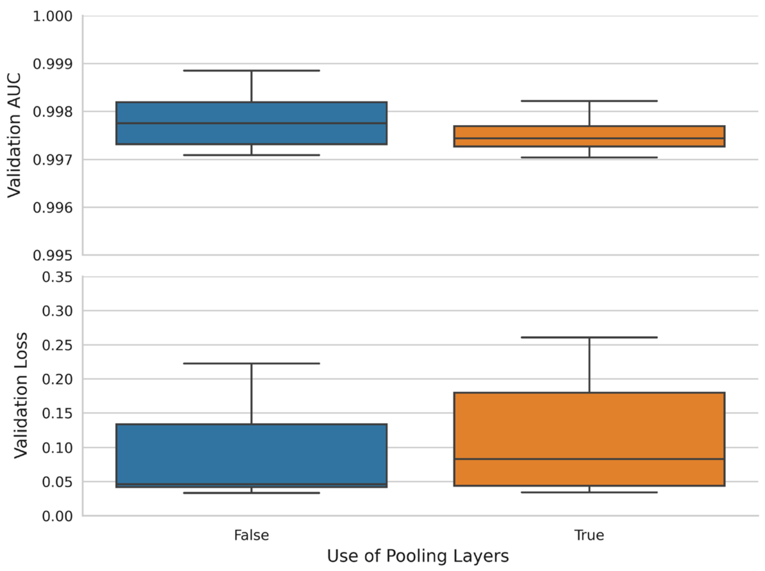

Fig. 10 Distribution of the validation AUC and loss for the 100 best trials, by use of pooling layers

models with or without max-pooling layers. Models without max-pooling seem to achieve better results in both metrics while being more consistent in terms of loss function values.

Taking into account the results of the hyperparameter optimization, we went on to validate our approach with the values shown in the last column of Table 5.

5.3 Player-based cross-validation

Previous works conducted experiments using rudimentary validation techniques and very small datasets.

Alayed et al. (2013) performed tenfold cross-validation in 7.6 h of data from only 2 players. Galli et al. (2011) used a simple train-test split with a dataset of also 2 players, but only 1 h of data. In Islam et al. (2020), interactions from 20 players were collected but there is no reference to the volume of their dataset. Yeung et al. (2006) collected a mere 1.7 h of data from 3 players and divided it into training, validation, and test datasets. As seen in Fig. 5, our dataset consists of roughly 490 h of gameplay and contains data from 128 players in total.

To quickly take action against fraudulent behavior, cheat detection systems must make accurate classifications for new players. No previous work took this necessity into account, so we developed a player-based cross-validation method to address it.

This player-based cross-validation method consists of training a model in all players except for one left out for validation. We describe this method in Algorithm 1.

Player-based cross-validation not only tests our approach's ability to perform well in unseen data but most importantly of detecting cheating interactions in unknown players.

Table 6 Results of player-based cross-validation in triggerbot detection

Player	Number of records	AUC	Best threshold *t*	FPR for *t*	TPR for *t*	Accuracy for *t*
1	40	0.997	0.080	0.047	1	0.975
2	375	1	0.381	0	1	1
3	121	0.996	0.027	0.010	0.95	0.992
4	129	1	0.54	0	1	1
5	126	1	0.348	0	1	1
7	26	1	0.591	0	1	1
8	320	0.992	0.005	0.027	1	0.978
Avg	162.429	0.998	0.282	0.012	0.993	0.992

Table 7 Results of player-based cross-validation in aimbot detection

Player	Number of records	AUC	Best threshold *t*	FPR for *t*	TPR for *t*	Accuracy for *t*
1	29	1	0.049	0	1	1
2	381	0.954	0.160	0.011	0.895	0.984
3	109	0.996	0.073	0.030	1	0.972
4	133	0.991	0.124	0.032	1	0.970
5	110	1	0.214	0	1	1
6	25	1	0.225	0	1	1
7	29	1	0.054	0	1	1
8	314	0.995	0.225	0.016	1	0.984
Avg	137.625	0.992	0.141	0.011	0.987	0.989

Algorithm 1: Player-Based Cross-Validation.

```
$\boldsymbol{X} \leftarrow$ dataset of multivariate time series
$\boldsymbol{y} \leftarrow$ labels for each example in the dataset
$\boldsymbol{players} \leftarrow$ list of players with records in
for $p$ in $\boldsymbol{players}$ do
    $\boldsymbol{X}_p \leftarrow$ records of player $p$
    $\boldsymbol{y}_p \leftarrow$ labels for player $p$
    $\boldsymbol{X}_o \leftarrow$ records of other players
    $\boldsymbol{y}_o \leftarrow$ labels of other players
    $model \leftarrow$ new instance of proposed CNN architecture
    Train($model$, $\boldsymbol{X}_o$, $\boldsymbol{y}_o$)
    Evaluate($model$, $\boldsymbol{X}_p$, $\boldsymbol{y}_p$)
end
```

We repeated this experiment for both cheats in our dataset: triggerbot and aimbot. The results are presented in Tables 6 and 7 respectively. Triggerbot results were better, which might be due to the patterns associated with this type of cheat being easier to detect or the fact that there is more data on this type of cheat.

We chose the best by maximizing the validation true positive rate (TPR), as long as the false positive rate (FPR) didn't exceed 5%, which did not occur for any instance of our experiment.

The left column in Fig. 11 shows the validation receiver operating characteristics curve for aimbot detection on four players. The right column shows the models' output distribution according to the ground truth. We can see that our models produce a very clear distinction between fraudulent and legitimate interactions.

The magnitude of the predictions varied mostly due to the stochastic nature of the model's parameter initialization.

6 Result discussion

Results show that our models were able to establish a clear distinction between legitimate and fraudulent gameplay.

As previously mentioned, max-pooling can enable learning longer patterns, and according to our hyperparameter search, it allows for deeper networks. Different games or applications may require this additional capacity. For each new domain where this approach is applied, a new hyperparameter search should be performed to find a well-performing architecture.

The presented cross-validation method allowed us to show that our models learn patterns that are not player-specific. Our models can detect fraudulent players even if they weren't exposed to their behavior. We suspect that the principal source of variation in our results is the fact that some of the tested players (such as player #2) represent a significant portion of our dataset.

We used a reduced set of event types (movement keys, mouse buttons, and mouse movements). The number of necessary event types to include in the multivariate time series might vary for different domains, which influences the architecture resulting from the hyperparameter search.

Our models achieve higher accuracy than reported in previous related work (Galli et al., 2011; Alayed et al., 2013; Islam et al., 2020; Alkhalifa, 2016) while also being tested in a much larger dataset. Although we're not able to test these previous approaches with our data (since they use completely different structures and in-game data), we argue that by comparing the reported accuracy we can safely suggest that our work constitutes an improvement at this task.

The ability to analyze player behavior based solely on peripheral input data is remarkable, allowing for the application of this system in many different video games, perhaps with different input methods, thus greatly reducing the need for manual feature engineering. This approach can even translate to other tasks related to modeling human activity, beyond video games.

This dataset has a large volume of data, collected from a wide variety of players in a real-world context. In this sense, we are confident that these results will translate to (or even improve with) communities with thousands or even millions of players.

7 Conclusion and future work

In this work, we developed a ground-breaking approach to cheat detection in video games. We applied deep learning and multivariate time series to human–computer interaction data to capture behavioral patterns in cheater mouse movements and keystrokes.

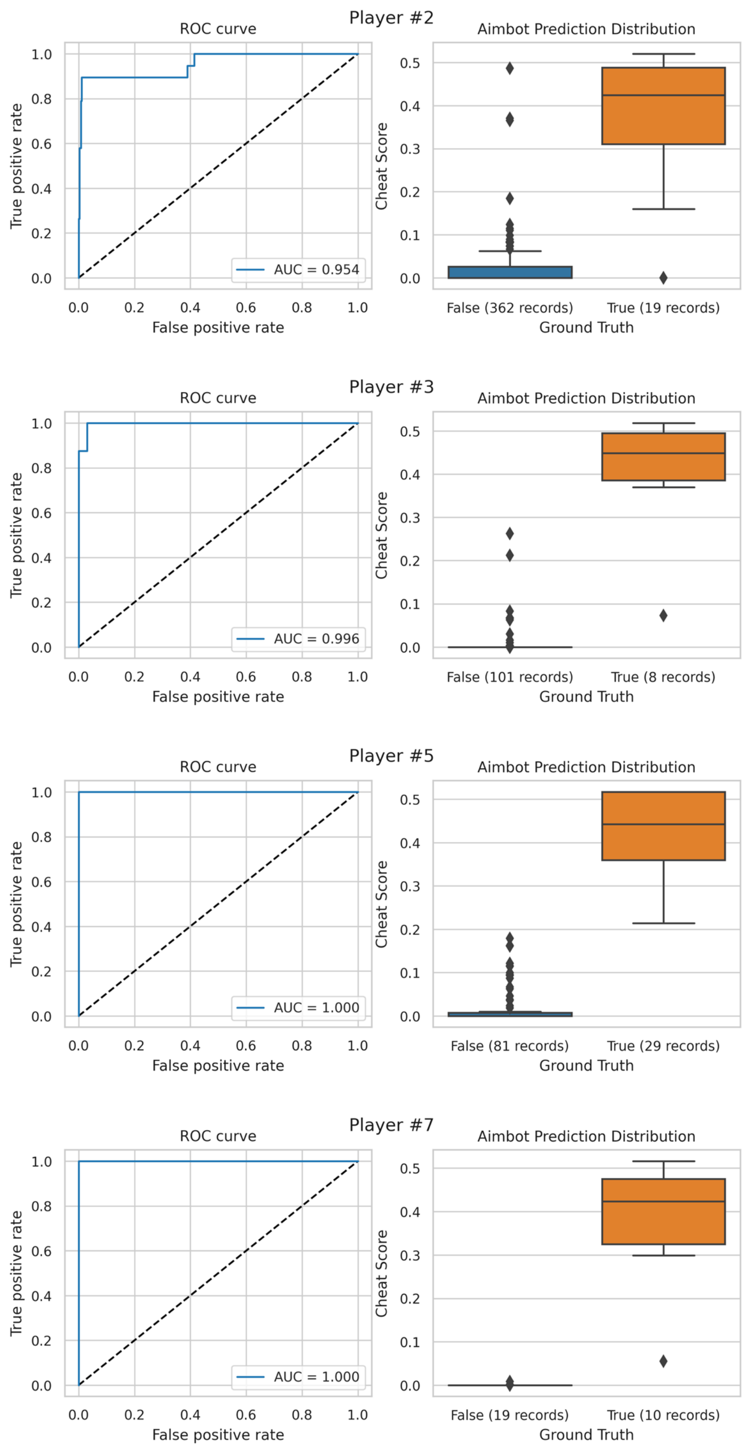

Fig. 11 Validation AUC and prediction distribution for 4 of the tested players

Our models showed results that are extremely positive when compared to any previous work. Additionally, our approach can be applied to any game, any input method, and perhaps other use-cases involving human activity.

This work provides a starting point for a unified framework for using deep learning in video games and should prove useful to the video-game industry, which is in great expansion and requiring such technologies.

For future work, testing this approach with a variety of video games and larger communities is a priority. Another path to be explored is to automate the model architecture design even further by adopting a more sophisticated hyperparameter search.

Another important path to investigate is the use of unsupervised methods to leverage great volumes of unlabeled data, not just in cheat detection but also in other use-cases such as grouping players in clusters based on their experience.

Finally, we envision the core ideas in this approach extrapolating beyond video games and finding uses in HCI applied to wellbeing.

Funding Not applicable.

Availability of data and material Not applicable.

Declarations

Conflict of interest Not applicable.

Code availability Not applicable.

Additional declarations Not applicable.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

Abadi, M.,, Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz R, Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S, Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org/, software available from tensorflow.org

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019) Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD international conference on knowledge discovery and data mining*.

Alayed, H., Frangoudes, F., & Neuman, C. (2013). Behavioral-based cheating detection in online first person shooters using machine learning techniques. In *2013 IEEE conference on computational inteligence in games (CIG)* (pp. 1–8). https://doi.org/10.1109/CIG.2013.6633617

Alkhalifa, S. (2016). Machine learning and anti-cheating in fps games. Ph.D. thesis, University College London. https://doi.org/10.13140/RG.2.2.21957.86242

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2018) Conditional time series forecasting with convolutional neural networks. arXiv:1703.04691

Breiman, L. (1996). Bagging predictors. *Machine Learning, Morgan Kaufmann, 24,* 123–140. https://doi.org/10.1023/A:1018054314350.

Carneiro, D., Pimenta, A., Gonçalves, S., Neves, J., & Novais, P. (2016). Monitoring and improving performance in human–computer interaction. *Concurrency and Computation: Practice and Experience, 28*(4), 1291–1309.

Chollet, F., et al. (2015) Keras. https://keras.io

Cui, Z., Chen, W., & Chen, Y. (2016). Multi-scale convolutional neural networks for time series classification. arXiv:1603.06995

Cun, L., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1990) Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems, Morgan Kaufmann* (pp. 396–404).

Dietterich, T. G. (2002). Machine learning for sequential data: A review. In T. Caelli, A. Amin, R. P. W. Duin, D. de Ridder, & M. Kamel (Eds.), *Structural, syntactic, and statistical pattern recognition* (pp. 15–30). Springer.

Du, S., Li, T., & Horng, S. (2018) Time series forecasting using sequence-to-sequence deep learning framework. In *2018 9th international symposium on parallel architectures, algorithms and programming (PAAP)* (pp. 171–176). https://doi.org/10.1109/PAAP.2018.00037

Filonov, P., Lavrentyev, A., & Vorontsov, A. (2016) Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model. arXiv:1612.06676

Galli, L., Loiacono, D., Cardamone, L., & Lanzi, P. L. (2011) A cheating detection framework for unreal tournament iii: A machine learning approach. In *2011 IEEE conference on computational intelligence and games (CIG'11)* (pp. 266–272). https://doi.org/10.1109/CIG.2011.6032016

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org

Islam, M. S., Dong, B., Chandra, S., Khan, L., & Thuraisingham, B. M. (2020). Gci: A gpu based transfer learning approach for detecting cheats of computer game. *IEEE Transactions on Dependable and Secure Computing*. https://doi.org/10.1109/TDSC.2020.3013817.

Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2018). Lstm fully convolutional networks for time series classification. *IEEE Access, 6,* 1662–1669. https://doi.org/10.1109/ACCESS.2017.2779939.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:14126980

Lin, S., Clark, R., Birke, R., Schönborn, S., Trigoni, N., & Roberts, S. (2020) Anomaly detection for time series using vae-lstm hybrid model. In *ICASSP 2020–2020 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 4322–4326). https://doi.org/10.1109/ICASSP40776.2020.9053558

Liu, C., Hsaio, W., & Tu, Y. (2019). Time series classification with multivariate convolutional neural network. *IEEE Transactions on Industrial Electronics, 66*(6), 4788–4797. https://doi.org/10.1109/TIE.2018.2864702.

Lu, W., Cheng, Y., Xiao, C., Chang, S., Huang, S., Liang, B., & Huang, T. (2017). Unsupervised sequential outlier detection with deep architectures. *IEEE Transactions on Image Processing, 26*(9), 4321–4330.

Mehdiyev, N., Lahann, J., Emrich, A., Enke, D., Fettke, P., & Loos, P. (2017). Time series classification using deep learning for process planning: A case from the process industry. *Procedia Computer Science,114*, 242–249. https://doi.org/10.1016/j.procs.2017.09.066, https://www.sciencedirect.com/science/article/pii/S1877050917318707, complex Adaptive Systems Conference with Theme: Engineering Cyber Physical Systems, CAS October 30–November 1, 2017, Chicago, Illinois, USA

Mehtab, S., & Sen, J. (2020). Stock price prediction using convolutional neural networks on a multivariate timeseries. arXiv:2001.09769

Pao, H., Chen, K., & Chang, H. (2010). Game bot detection via avatar trajectory analysis. *IEEE Transactions on Computational Intelligence and AI in Games, 2*(3), 162–175. https://doi.org/10.1109/TCIAIG.2010.2072506.

Pimenta, A., Carneiro, D., Novais, P., & Neves, J. (2014). Analysis of human performance as a measure of mental fatigue. In J. S. Pan, M. Woźniak, H. Quintian, & E. Corchado (Eds.), *Polycarpou M, de Carvalho ACPLF. Hybrid artificial intelligence systems*. Berlin: Springer.

Pimenta, A., Carneiro, D., Neves, J., & Novais, P. (2015). Improving user privacy and the accuracy of user identification in behavioral biometrics. In P. Machado, E. Costa, A. Cardoso, & F. Pereira (Eds.), *Progress in artificial intelligence*. Springer.

Sagheer, A., & Kotb, M. (2019). Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing, 323,* 203–213. https://doi.org/10.1016/j.neucom.2018.09.082.

Siegel, B. (2020). Industrial anomaly detection: A comparison of unsupervised neural network architectures. *IEEE Sensors Letters, 4*(8), 1–4. https://doi.org/10.1109/LSENS.2020.3007880.

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*(1), 1929–1958.

Tan, H. X., Aung, N. N., Tian, J., Chua, M. C. H., & Yang, Y. O. (2019). Time series classification using a modified lstm approach from accelerometer-based data: A comparative study for gait cycle detection. *Gait and Posture, 74,* 128–134. https://doi.org/10.1016/j.gaitpost.2019.09.007.

Wan, R., Mei, S., Wang, J., Liu, M., & Yang, F. (2019). Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. *Electronics, 8,* 8.

Wang, Z., Yan, W., & Oates, T. (2017) Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (IJCNN)* (pp. 1578–1585). https://doi.org/10.1109/IJCNN.2017.7966039

Xing, Z., Pei, J., & Keogh, E. (2010). A brief survey on sequence classification. *SIGKDD Explorations Newsletter, 12*(1), 40–48. https://doi.org/10.1145/1882471.1882478.

Yeung, S. F., Lui, J. C. S., Jiangchuan, L., & Yan, J. (2006) Detecting cheaters for multiplayer games: theory, design and implementation[1]. In *CCNC 2006. 2006 3rd IEEE consumer communications and networking conference, 2006*, (Vol .2, pp. 1178–1182). https://doi.org/10.1109/CCNC.2006.1593224

Zhao, B., Lu, H., Chen, S., Liu, J., & Wu, D. (2017). Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics, 28*(1), 162–169. https://doi.org/10.21629/JSEE.2017.01.18

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2016). Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Frontiers of Computer Science, 10,* 96–112. https://doi.org/10.1007/s11704-015-4478-2.