2024 IEEE International Joint Conference on Biometrics (IJCB) | 979-8-3503-6413-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/IJCB62174.2024.10744488

# Useckit: An Open-Source Deep-Learning Toolkit Bundling State-Of-The-Art Algorithms for Evaluating Behavioral Biometrics

Jonathan Liebers
University of Duisburg-Essen
Essen, Germany
jonathan.liebers@uni-due.de

Tristan Kley
University of Duisburg-Essen
Essen, Germany
tristan.kley@stud.uni-due.de

Carina Liebers
University of Duisburg-Essen
Essen, Germany
carina.liebers@uni-due.de

Uwe Gruenefeld
University of Duisburg-Essen
Essen, Germany
uwe.gruenefeld@uni-due.de

Stefan Schneegass
University of Duisburg-Essen
Essen, Germany
stefan.schneegass@uni-due.de

## Abstract

*There is an endeavor in the Human-Computer Interaction (HCI) community to create novel authentication schemes so that passwords finally become obsolete and practical security is enhanced. For this purpose, researchers combine behavioral biometrics with deep learning. However, because the process of creating neural networks is inherently complex and each model architecture has certain limitations, implementing research prototypes is a time-consuming and challenging task. Therefore, we present useckit, an open-source toolkit that provides deep learning algorithms to support the creation of scientific evaluations in authentication research. Useckit provides multiple paradigms for implementing user verification and identification, functions to calculate common metrics, and neural network architectures founded in literature. It is written in Python and supports researchers and practitioners in creating, implementing, and rigorously evaluating novel deep learning-based authentication schemes.*

## 1. Introduction

With the increasing number of personal devices people interact with in their daily lives, the number of authentications one performs per day is ever-increasing. Today, personal identification numbers (PINs) and passwords are still primarily necessary for this process, a situation that has already persisted for 60 years [2]. However, there exists a broad consensus that passwords are a predictable [17], imperfect technology [4] and that the increasing number of uses leads to fatigue [41], while the user is often regarded as the weakest link in the security chain [1, 4]. Also, one in ten password entries fails [6]. Thus, recent research has motivated the need for user-centric approaches to authentication.

Besides knowledge-based methods, like PINs and passwords, token- and biometric-based approaches to authentication exist. Token-based approaches function by the user possessing a token, such as a security key that is used for authentication. From a technical perspective, these are highly secure. However, the token must always be carried by the human user. Practical security is heavily impacted when the token is lost, as the token finder can easily impersonate the original owner. Instead, researchers recently focused increasingly on biometrics [21, 13], which encapsulates the means of authenticating a person by their true identity, through biometric traits that either the body has (e.g., a fingerprint) or through traits imposed by human behavior (e.g., specific gait patterns or voice). However, biometrics are sometimes criticized for being less secure than their established alternatives. The argument is that the computer that recognizes a human by the biometric pattern can fail or be tricked, possibly granting access to an untrusted third party [14, 31].

While the argument of biometrics being theoretically less secure can undoubtedly be made, the sensing and computing capabilities of computing devices increase strongly as new devices incorporate an increasing number of sensors. Also, innovations in artificial intelligence and deep learning lead to the boundary of biometric recognition being constantly extended [44], as new approaches overcome previously established barriers [8], due to their automated feature engineering and computational power. As a result, biometric authentication moved into the focus of researchers [2]. Biometrics particularly consider human factors, as they can relieve users of the burden of authentication, mainly when the created schemes occur through an implicit interaction [43, 20].

However, creating deep learning-based biometric authentication schemes is still challenging. On the one hand, two biometric modes exist for authenticating users, "verification" and "identification", which define the functionality of the authentication scheme [19]. Each mode has a set of specific metrics associated with it that researchers need to report during evaluation, ranging from accuracy, F1-score, and confusion matrices to equal error rates and receiver-operating characteristics. Next to the reporting, the implementation itself also poses some challenges. Here, various approaches exist to implement biometric matching modules, like machine learning or deep learning methods. These partly require special pre-processing functions such as window slicing techniques [7, 24], which are complicated to implement. Last but not least, scalability is an issue, as deep learning models are costly to train since certain approaches require re-training when a single, new identity is added to the system.

To facilitate the creation of novel deep learning-based authentication schemes, we introduce *useckit*. Written in the Python programming language, useckit is an open-source software library[1] that integrates well with the established data-science ecosystem and connects deep learning paradigms with the automated calculation of security-related evaluation metrics. Further, it provides specialized pre-processing functions such as window slicing. We thereby provide a free and extensible toolkit that facilitates the creation of new authentication schemes, employing state-of-the-art deep learning algorithms and model architectures to support researchers in conducting their evaluation. Additionally, the design of useckit allows for mitigating limitations that specific, popular approaches have, which are mainly related to the areas of closed-set and open-set identification, supporting rigorous research.

[1]Useckit and its source code can be retrieved from https://useckit.behavioral-biometrics.org.

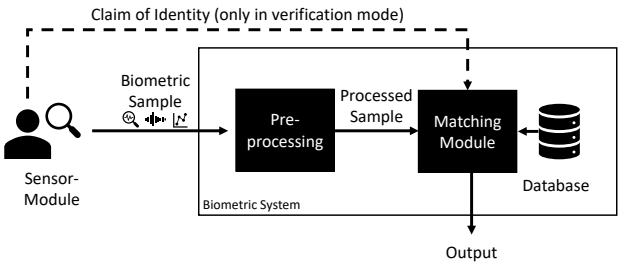


Figure 1. Biometric Authentication Systems as described by Jain et al. [19]. In any case, a user provides a biometric sample through a sensor module. The system then processes the sample, and the matching module provides an output. Deep learning models are often used to implement the matching module. Table 1 provides information on the system's input and output. The dashed arrow conveys a claim of identity and exists only in verification mode systems.

## 2. Creating Biometric Authentication Systems with Deep Learning

Following the description of Jain et al., biometric authentication systems (cf., Figure 1) are composed of a sensor module, a preprocessing function, a matching module, and a database [19, 18]. The sensor module is an interface that takes a biometric sample from the user. Next, the sample undergoes a quality assessment and other transformations in the preprocessing step, such as specialized feature extractions. The processed sample is moved to the matching module, which generates an output using information from the database. The output is determined by the biometric authentication system's mode, which is either set to "verification" or "identification" [19, 18].

### 2.1. Implementing a Matching Module with Deep Learning

Using deep neural networks in the biometric matching module is desired for two reasons. First, they can automate the feature extraction through their learning process, and thus, they can exceed manually crafted features, leading to higher accuracies [28]. Second, certain deep learning paradigms, such as models utilizing distance learning, enable high scalability. Such models are able to generalize to data that was unseen during the training phase; hence, they can be trained on one data set and applied to other data [42, 25]. Depending on the biometric authentication system's mode, the matching module receives different inputs and provides varying outputs, and each is specific to the deep learning-based matching module (cf., Table 1).

#### 2.1.1 Verification Mode

In verification mode, the system first receives a claim of identity (cf., Figure 1), telling the system which user account will be subject to the verification process. The system queries the internal database of stored biometric samples to find the reference sample for the desired user account [19]. Next, after preprocessing, the matching module decides whether the provided biometric sample is sufficient to grant the user access. Notably, the samples in the database had to undergo the same preprocessing as the matched (tested) sample. Here, an often utilized paradigm is to compute a similarity measure between the reference sample and the provided sample and to check whether the similarity falls below a threshold. Users are accepted if the computed similarity falls below the threshold value and rejected if the value exceeds the threshold. Table 1 lists applicable deep learning approaches.

#### 2.1.2 Identification Mode

In the identification mode, the matching module does not receive a claim of identity. Instead, it needs to find the correct account of the user only *by* their provided biometric sample [19]. To do so, many recent academic works used classifiers for this task [38, 27, 28, 26, 23, 32, 29]. However, for identification, classifiers are limited in two ways: first, the classifier must be retrained upon adding a new class to the system. Second, a classifier will classify all received data according to its trained classes, even if the data is an (extreme) outlier, such as an adversary attacking the system. This is due to classifiers being unable to meaningfully reject any data, which is a severe limitation for biometrics. Thereby, one must distinguish between open-set and closed-set identification, where the latter is implemented in many works. A biometric system will always accept all users using classifiers as their matching module; however, it might assign the wrong classes. In contrast, open-set classification can reject outliers and assign classes when it finds the data to be familiar [3]. To implement open-set identification, distance functions and autoencoders can be used, as they can query the database for the most similar sample. Through a threshold, they can also perform rejections, enabling open-set identification that classifiers can only hardly implement.

### 2.2. Scalability

Scalability is a property depending on the model type. It becomes relevant when a new user is added to the biometric system. Depending on the approach, none, one, or all involved deep learning models need to be retrained after a new insertion into the biometric system's database, which can cost significant resources. Table 1 provides an overview. For instance, classifiers must always be retrained when adding a new class to the biometric system. In the case of autoencoders, only one new model must be trained: the model associated with the new person [9]. Noteworthy is the role of the distance function, as no new model needs to be trained here [42]. If a distance function can generalize the aspects of similarity in a biometric sample, it can generalize this knowledge to completely unseen data. This allows for high scalability, even in a global setting [25]. Recently, neural networks have been combined with distance functions (such as the Euclidean distance) to estimate the similarity between two input samples [44]. Therefore, the neural network helps provide a number that describes the similarity between two biometric samples, where a lower number indicates semantic similarity and a higher number indicates semantic dissimilarity. This principle is called "distance metric learning" [36, 22, 30]. Recent works by Miller R. et al. and Schell et al. [42, 34] have shown the applicability in behavioral biometrics. One of the main benefits is scalability, as deep metric learning can generalize to unseen data based on the created embedding. Thereby, the training data of the system and the enrollment data can be different, which is virtually impossible for classifiers. Instead, the embedding and the distance function can be extracted and used with previously unseen data, as no recreation of the embedding is necessary to deal with new user identities [42].

## 3. Useckit

Useckit provides methods to implement a matching module in biometric authentication systems and supports the researcher in creating an evaluation by providing methods to compute widely used biometrics-specific key metrics. It creates a hierarchical folder structure that contains serialized deep learning models, numerical metrics in text and JSON format, and plots, being human and machine-readable at the same time. Useckit is available for download via PyPI[2].

### 3.1. Data set: Training, Validation, Test, Enrollment, and Matching

Useckit requires the user to provide their data in a data set object. Besides defining a training-, validation- and test set with which the models are trained, users can also provide an enrollment and matching set, as these can differ. The first test data set corresponds to the enrollment data of the trained

[2]Useckit can be installed from the python package index (PyPI) via `$ pip install useckit` (cf., https://pypi.org/project/useckit/#files).

Table 1. Overview of biometric authentication modes and their implications when employing algorithms such as classifiers, distance functions (Distance Func.), or autoencoders (AutoEnc.) in the matching module. Input and output are related to the system's context (cf., Figure 1). The base chance describes the random guess probability for $N$ classes, i.e., identities. ✓ symbolizes that the algorithm is suitable or unsuitable (×) for the given mode. The second symbol indicates scalability when a new identity is enrolled: ☒ symbolizes that all created deep learning models need to be retrained, ⊡ that only one model needs to be trained for the added class, and ⊟ signalizes that no model needs to be retrained (i.e., ideal scalability).

| Biometric Auth. Mode | Input | Output | Base Chance | Classifiers | Distance Func. | AutoEnc. |
|---|---|---|---|---|---|---|
| Verification | Claim of Id., sample | Accept / Reject | 1/2 | ✓ ☒ | ✓ ⊟ | ✓ ⊡ |
| Identification (closed-set) | Only sample | Identity | 1/N | ✓ ☒ | ✓ ⊟ | ✓ ⊡ |
| Identification (open-set) | Only sample | Identity / Reject | 1/(N+1) | × | ✓ ⊟ | ✓ ⊡ |

system, stored in its database (cf., Figure 1), and the second data set symbolizes the matching data provided to be used in the testing phase. Here, it is possible to set both the training and enrollment data to the same object. However, cases might exist where this is not desired (e.g., in deep metric learning). For example, distance functions can generalize to data not seen during training. Each data set is specified through two arrays, one for the data and one for the labels. The labels should be provided as strings or integers and are automatically transformed into special forms by useckit when required by the model (e.g., one-hot encodings will automatically be applied if required by the model). Lastly, useckit supports using k-cross evaluation through its data set class if insufficient data exists for a hold-out validation style [39].

## 3.2. Preprocessing Functions

Useckit deliberately provides only a limited set of preprocessing functions, such as window-slicing, majority voting, and essential data checks, as preprocessing often requires domain-specific functions, depending on the biometric modality.

### 3.2.1 Window Slicing and Majority Voting

Useckit provides integrated support for window slicing. Window slicing is a technique that takes a biometric sample and creates new samples from it by applying a sliding window of size $s$, moving at a stride $w$ [24]. The window slides along the primary axis of the data, which is usually the time axis in behavioral biometrics, and extracts each segment of the original sample into a new sample, potentially increasing system accuracy [24]. Through window slicing, the number of samples can drastically increase, potentially providing more data for the deep-learning models to learn from while shortening the individual timespan of each sample. Users are warned if any issues with the window slicing parameters appear (e.g., the creation of incomplete slices). Special measures must be taken when evaluating samples originating from a window-slicing technique. It needs to be kept in mind that the slices are not original material that, for example, a participant in a user study has provided. If, for example, $n$ biometric samples exist that were transformed into $m$ biometric samples during window slicing, then the evaluation would usually cover the question of how often, given $n$ biometric samples, a user can be recognized through either verification or identification. Therefore, it is essential to transform these intermediary $m$ samples back to the original $n$ samples. Useckit provides automated support by performing this retransformation, utilizing majority voting functions by predicting each sample individually and then deriving a consensus from all predictions for each original sample.

### 3.2.2 Data Checks

Useckit automatically checks specific requirements imposed on the user's data, such as if the data was normalized correctly, e.g., if its values fall into the interval of $[-1; +1]$, and notifies the user if this is not the case. Also, it tests if "not a number" (NaN) values are in the data, which can lead to the deep learning models becoming unable to learn. Such values can be easily overlooked; useckit informs the user to prevent this.

## 3.3. Evaluation Paradigms, Models, and Methods

Useckit currently provides three different paradigms that are derived from previous work to enable verification and identification schemes. They are i) time-series classification models (with categorical or binary cross-entropy loss) [29, 32, 28, 15], ii) distance metric learning models (with loss functions estimating similarity) [42, 34, 30], and iii) anomaly-detection models (with loss functions based on sample reconstruction error) [9]. These paradigms are to be understood as a means for creating a usable security evaluation and are comparable to choosing different system design approaches. Metrics from the final system may vary, with useckit reporting the applicable ones.

### 3.3.1 Time-Series Classification Models

The models for time-series classification are adapted from the survey on deep learning for time-series clas-

sification by Fawaz et al. [15] who released their source code on Github[3]. These models were previously adapted from literature and were already tested in usable-security evaluations [32, 28, 29] to a varying extent of accuracy. The model architectures embedded in useckit, which are based on the previous work of Fawaz et al. [15] are i) a Time Convolutional Neural Network (Time-CNN) [48], ii) Multi-Layer Perceptron (MLP) [47], iii) Fully Convolutional Neural Network (FCN) [47] (in two variants with "same" and "valid" padding), iv) Residual Network ResNet [47], v) Encoder [45], vi) Multi-scale Convolutional Neural Network (MCNN) [10], vii) Time Le-Net t-LeNet [24], viii) Multi-Channel Deep Convolutional Neural Network (MCDCNN) [49], ix) Time Warping Invariant Echo State Network (TWIESN) [46], and x) InceptionTime (Inception) [16]. All classification models undergo a supervised learning process in their training phase and must be retrained if a new identity is added to the system.

#### 3.3.2 Distance Learning Models

Distance learning is performed through three different Siamese models in useckit. Through supervised learning, siamese networks implement a distance function similar to Euclidean or cosine distance functions, yet the features are crafted by the neural network in an embedding space. This means that siamese models can be used to determine the similarity between two samples on a numerical scale, where 0 indicates equality between two samples and values larger than 0 denote an increasing difference in similarity between two samples. Siamese networks have been introduced by Bromley et al. for the task of hand-written signature verification [5] and are also used for authentication [34]. For this, we provide a siamese network with a contrastive loss following the design and implementation of Mehdi [33]. The contrastive network is based on an offline pair-generation function that creates pairs (2-tuples) of samples, where pairs are generated from the data set and are automatically annotated by useckit, whether they belong to the same class or not. Two samples from the same class are considered similar, hence labeled with a 0, whereas two samples from different classes receive a 1 as a label. In the training phase, the network will try to extract knowledge of where the within-class similarity between samples is located and where differences exist in an intra-class setting. This enables an estimation of similarity. Further, we provide two default Siamese models that use triplet loss.

[3]Hassan Ismail Fawaz. Deep Learning for Time Series Classification on GitHub. https://github.com/hfawaz/dl-4-tsc, last retrieved July 15, 2024.

Listing 1. Low-level usage of Useckit, where a custom model description is provided. The parameterization of the internal API enables flexibility. The function "my_model" creates a model architecture that is passed to TSCParadigm, which then invokes training and evaluation functions.

```
# Note: imports are omitted.

# `evaluation_methods` receives a list of objects
# that follows an interface to implement custom
# evaluation methods. `model_description` takes a
# custom function that returns an architecture.

def my_model(input_layer: keras.layers.Input,
             nb_classes: int) -> \
            (keras.models.Model, list[Callable]):
  layer_1 = keras.layers.Flatten()(input_layer)
  # ... (additional model layers)
  layer_last = keras.layers.Dense(nb_classes,
               activation='softmax')(layer_1)
  model = keras.models.Model(inputs=layer_1,
                             outputs=layer_last)
  model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model

tsc = TSCParadigm(
  evaluation_methods=[TSCIdentification()],
  prediction_model=TSCKerasModel(
    model_description=my_model, nb_epochs=1000)
  )
tsc.evaluate(dataset)  # run evaluation
```

Schroff et al. introduced triplet loss for biometric face verification tasks [44]. It extends the contrastive loss by creating 3-tuples from the given data, where an anchor sample is paired with a positive and negative sample. Positive samples originate from the same class as the anchor sample, and negative samples come from a different class. We follow the implementation of Morgan that uses online learning and utilizes the triplet loss functions provided in the TensorFlow addons [35]. Additionally, we provide an offline learning variant, following the design of Essam and Valdarrama [12]. This model type can generalize to entirely different data not seen during training. Therefore, the training and enrollment data can differ for the respective purpose. It is not required to retrain distance learning models to add new identities to the system, and they are particularly renowned for their scalability [25].

#### 3.3.3 Anomaly-Detection Models

Anomaly Detection takes place through variational autoencoder models [11], as demonstrated by Chen et al. [9]. For each identity, i.e., for each class in the data set, useckit automatically trains one autoencoder. After training, the autoencoder receives a new sample and tries to reconstruct it.

Listing 2. Python code to create a data set in useckit and to run a high-level identification evaluator. A distance learning evaluator facilitating a Siamese neural network and a time series classification model are created [5, 15].

```
import numpy as np
import useckit
from useckit.Evaluators import \
  TSCEvaluator, DistanceLearningEvaluator

# `DATA` and `LABELS` contain the original data
x_train: np.ndarray = DATA[SESSION == 1]
y_train: np.ndarray = LABELS[SESSION == 1]
x_test: np.ndarray = DATA[SESSION == 2]
y_test: np.ndarray = LABELS[SESSION == 2]

# Create a useckit dataset object.
DATASET =  useckit.Dataset(
    trainset_data = x_train,
    trainset_labels = y_train
    testset_enrollment_data = x_train,
    testset_enrollment_labels = y_train,
    testset_matching_data = x_test,
    testset_matching_labels = y_test)
verbose: bool = False # sets output verbosity
epochs: int = 1000 # number of training epochs

# runs default distance learning models
dle = DistanceLearningEvaluator(DATASET,
                                epochs=epochs,
                                verbose=verbose)
dle.evaluate()

# runs default time series classification models
tse = TSCEvaluator(DATASET,
                   epochs=epochs,
                   verbose=verbose)
tse.evaluate()
```

As a result, the success of the reconstruction can be measured by the mean square error in the reconstruction. It has been shown that reconstruction works better for the class that the autoencoder was trained with and worse for other classes [40]. A limit of the acceptable amount of error in reconstruction can be set, shielding the trained class from anomalies that fall outside this class. In contrast to other methods, autoencoders utilize unsupervised learning. A new autoencoder needs to be trained to add a new identity to the system.

#### 3.3.4 Extensibility for Custom Architectures

Users can also provide their own model architectures to run within useckit. Every of Useckit's three paradigms inherits from "ParadigmBase" and can be provided a prediction model and an array of evaluation methods. The prediction model inherits from the abstract base class "PredictionModelBase", which provides an interface that needs to be implemented for methods to fit its model and to perform a prediction, as well as for serialization and deserialization to and from disk. Furthermore, the evaluation methods all derive from an abstract base class "EvaluationMethodBase", and implementing its "evaluate"-method allows for execution by useckit. Custom models can be provided as functions passed to useckit (cf., Figure 2 and Listing 1).

### 3.4. Evaluation Methods

Useckit provides three different evaluation methods for i) verification, ii) closed-set identification (identification without reject), and iii) open-set identification (identification with reject). The identification methods automatically provide reports on the classification accuracy, reporting metrics such as accuracy, macro average, and weighted average over all classes, as well as precision, recall, and F1-score. The reports are saved as plain text files to disk and in machine-readable JSON format, which can be used for statistical analysis of the deep learning model's performance if desired by the researcher. Such JSON files can easily be imported to other tools, such as R, that are desired for their statistics ecosystem. Listing 3 provides an overview of a generated directory's structure. For the verification mode metrics, useckit provides accuracy, precision, recall, sensitivity, and specificity calculations. Additionally, useckit enables calculations of the equal error rate (EER), receiver operating characteristic (ROC), and area under the ROC curve (AUC / AUROC). Users can also specify custom evaluation methods to calculate specialized metrics through the API.

Listing 3. Hierarchical filesystem structure generated by useckit, containing a folder per model with classification reports as text and JSON files, serialized deep learning models as hdf5-files, execution timings, and PDF plots.

```
useckit_out_<timestamp>
 ├── experiment_1_DistanceMetricParadigm_-
 │   ContrastiveKerasPredictionModel
 │  ├── evaluation_identification
 │  │  ├── classification-report.json
 │  │  ├── classification-report.txt
 │  │  └── confusion-matrix.pdf
 │  ├── keras_pred_model_out
 │  │  ├── best_model.hdf5
 │  │  ├── history_acc.pdf
 │  │  ├── history_loss.pdf
 │  │  ├── last_model.hdf5
 │  │  └── model_init.hdf5
 │  ├── paradigm_experiment_1_-
 │  │   DistanceMetricParadigm_-
 │  │   ContrastiveKerasPredictionModel.pkl
 │  └── timing.txt
 ├── experiment_2_[...]
 │  └── [...]
[...]
```

### 3.5. High-Level API

For each of the paradigms, useckit also offers a high-level API. Instead of creating and customizing the paradigms individually, all of useckit's pre-defined models can be executed simultaneously. For this case, useckit provides a "TSCEvaluator" that automatically evaluates all included time-series classification models, a "DistanceLearningEvaluator" that automatically runs the contrastive loss and online triplet loss models, and an "AnomalyDetectionEvaluator", automatically training and evaluating the approach realized through variational autoencoders. Here, useckit provides default models and hyperparameters. Listing 2 provides an example of creating a data set and running an evaluation with the "TSCEvaluator". The design of the classes is kept intentionally simple so that the library user can provide it with their preprocessed data and labels and receive a broad impression of all approaches. At last, an "AutoEvaluator" exists, which runs all evaluators for the three paradigms. However, due to the range of tested approaches in the high-level API, the execution can take considerable computational time. GPU execution is supported and substantially reduces the required time.

## 4. Case Study

In the following, we report on a case study in which we apply useckit[4]. To do so, we recreate a previous behavioral biometrics study with a publicly available data set on the internet [27].

### 4.1. Preparing the Data Set

The case study uses a data set that consists of behavioral biometric data elicited in virtual reality (VR). It contains behavioral data from 16 participants who interacted with eight different virtual user interfaces (e.g., pressing a virtual button) through hand tracking in VR. The data set contains the position and rotation data of each point of the hand model, resulting in 182 features per sample. In the following, we focus on the "button scene" where participants pressed a virtual button. This interaction was repeated 12 times per participant, in addition to a session-repetition on a second day.

### 4.2. Using Useckit

The case study consists of 7 steps. In the first two steps, we load essential libraries and download the data set from the internet. Then, in the third step, we load the tabular data from their tabulator-separated format from the disk into memory. Next, in the fourth step, we inspect the data by creating descriptive statistics, especially over the lengths of each sample. Each button press in virtual reality had a different length in milliseconds, ranging between 397 and 665 frames ($M = 458, SD = 31$) at a sampling rate of 72 Hz.

### 4.3. Preprocessing

For preprocessing, we create a coordinate transformation by subtracting all positional columns from the first row's value, therefore, centering the positions with regard to the initial value. By applying this transformation, we receive only the relative movements of the head-mounted display and remove any imposed bias that could be introduced by, e.g., participants standing at a distinct location in the tracking space. Then, we extract the actual button interactions. We do so by checking when the button reported in the events column that a press had started and when a subsequent release was initiated.

We remove all data before and after these points in time per sample, with an additional respective margin of 1 second to capture the movement of the hand to and from the button, i.e., before and after the press. Since all participants were right-handed, we discarded all data from their left hand and set all missing values to zero to fill gaps created by tracking loss. Also, we apply a min-max-normalization as the final step of preprocessing. In the sixth step, we provide this data with the labels to useckit by creating a data set object. We set the training, validation data, and enrollment test set to the data obtained in the study's first session and perform a hold-out validation by testing only with the data obtained from the study's second session by setting this to the test matching data.

### 4.4. Model and Results

In the seventh step, we create an evaluator for each paradigm and run their evaluation method. Particularly the TSCEvaluator creates results that are easily comparable to the results of the original study [27]. Here, ResNet reports the best accuracy, a closed-set accuracy of 51.56%, which is slightly better than the original publication's reported accuracy of 49% [27]. An overview of the generated files can be found in Listing 3, where the classification report generated by scikit-learn provides key metrics such as accuracy, precision, recall, F1-score, prevalence, and bias [37]. Furthermore, a confusion matrix as PDF is created, and the trained models are serialized to disk.

[4]The source code for the case study is available in the file "examples/useckit-high-level.ipynb" that is distributed as part of useckit's source code.

## 5. Conclusion

We introduce "useckit", a free and open-source library intended to help with evaluations in human-centered security, where researchers seek to create novel, deep learning-based schemes for personal authentication. Useckit provides vital support by implementing three paradigms that can be used for user verification, closed-set, and open-set identification tasks.
Furthermore, useckit encompasses classification networks together with two other approaches to distance learning and anomaly detection, which are all suitable for user authentication in previous works. Additionally, next to providing predefined models and approaches, useckit is flexible and extensible so researchers can quickly extend the provided interfaces. Depending on the biometric authentication system's mode, useckit generates key metrics for identification such as accuracy, F1-score, and confusion matrices, but it can also perform thresholding operations to generate equal-error-rate plots and receiver-operating characteristics curves for user verification. Hence, useckit contributes to the creation of novel academic works by facilitating the evaluation of such works, as shown in a case study on a real behavioral biometric data set.

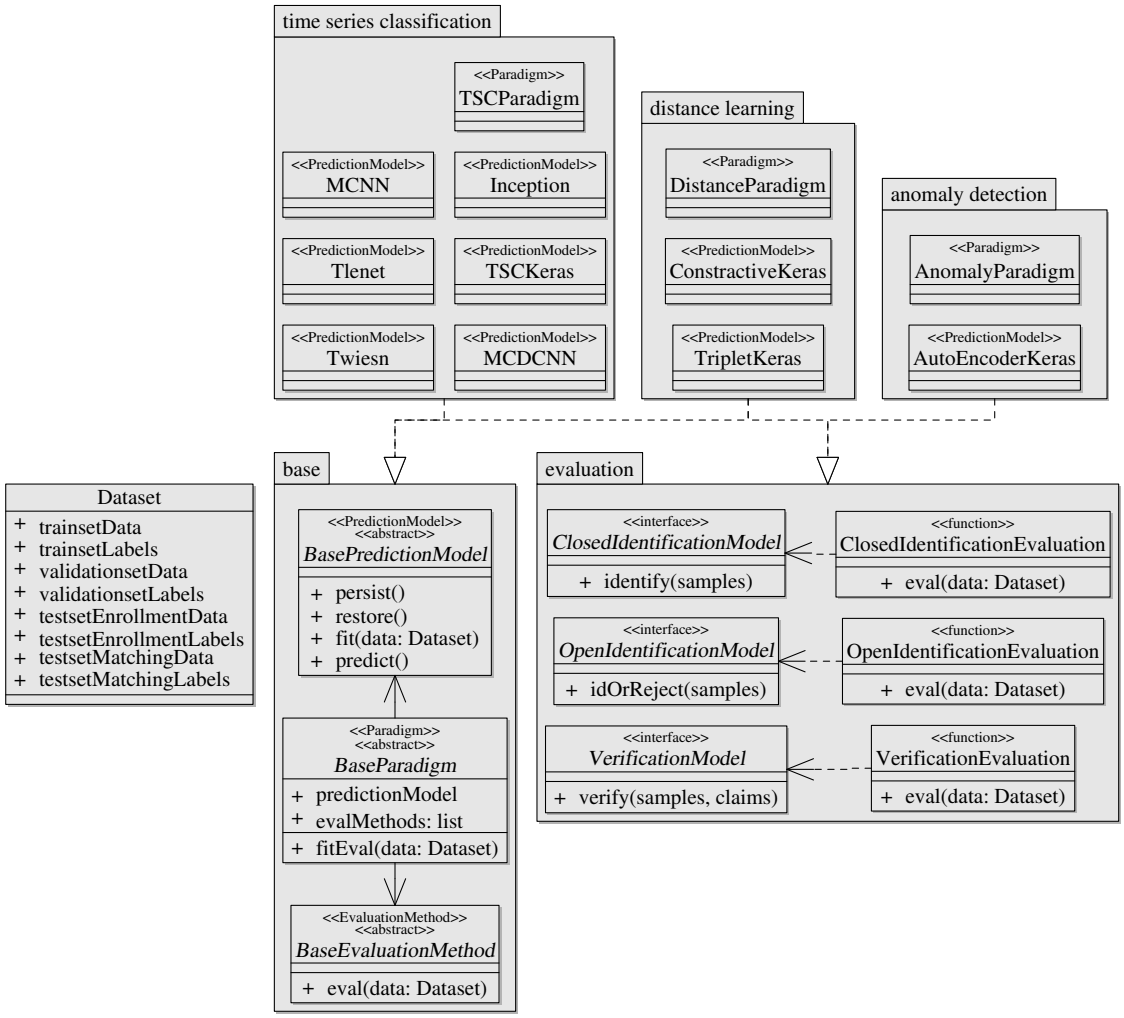


Figure 2. Unified Modeling Language (UML) class-diagram of the internal object-oriented structure of useckit, depicting the most important structures. The names of some functions have been shortened to increase readability.

## References


[1] A. Adams and M. A. Sasse. Users Are Not the Enemy. *Communications of the ACM*, 42(12):40–46, 1999.

[2] F. Alt and S. Schneegass. Beyond Passwords—Challenges and Opportunities of Future Authentication. *IEEE Security & Privacy*, 20(1):82–86, 2022.

[3] A. Bendale and T. E. Boult. Towards open set deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[4] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. Passwords and the evolution of imperfect authentication. *Commun. ACM*, 58(7):78–87, jun 2015.

[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature Verification using a "Siamese" Time Delay Neural Network. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993.

[6] S. Brostoff and M. A. Sasse. "Ten strikes and you're out": Increasing the number of login attempts can improve password usability. In *Workshop on Human-Computer Interaction and Security Systems at CHI 2003*, Fort Lauderdale, Florida, USA, 2003. ACM.

[7] A. Bulling, U. Blanke, and B. Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys*, 46(3):1–33, 2014.

[8] R. Cappelli, M. Ferrara, A. Franco, and D. Maltoni. Fingerprint verification competition 2006. *Biometric Technology Today*, 15(7):7–9, 2007.

[9] Y. Chen, Z. Yang, R. Abbou, P. Lopes, B. Y. Zhao, and H. Zheng. User Authentication via Electrical Muscle Stimulation. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 2021.

[10] Z. Cui, W. Chen, and Y. Chen. Multi-scale convolutional neural networks for time series classification, 2016. Available online at https://arxiv.org/abs/1603.06995, last retrieved on July 15, 2024.

[11] C. Doersch. Tutorial on variational autoencoders, 2016. Available online at https://arxiv.org/abs/1606.05908, last retrieved on July 15, 2024.

[12] H. Essam and S. Valdarrama. Image similarity estimation using a siamese network with a triplet loss, 2023. Available online at https://keras.io/examples/vision/siamese_network, last retrieved on July 15, 2024.

[13] A. Giaretta. Security and privacy in virtual reality – a literature survey, 2022. Available online at https://arxiv.org/abs/2205.00208, last retrieved on July 15, 2024.

[14] L. Hamid. Biometric technology: not a password replacement, but a complement. *Biometric Technology Today*, 2015(6):7–10, 2015.

[15] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[16] H. Ismail 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*, 34(6):1936–1962, Nov 2020.

[17] J. Bonneau. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, San Francisco, CA, USA, 2012. IEEE.

[18] A. K. Jain, P. Flynn, and A. A. Ross, editors. *Handbook of biometrics*. Springer, New York, NY, 2008.

[19] A. K. Jain, A. A. Ross, and K. Nandakumar, editors. *Introduction to Biometrics*. Springer US, Boston, MA, 2011.

[20] M. Jakobsson, E. Shi, P. Golle, and R. Chow. Implicit Authentication for Mobile Devices. In *Proceedings of the 4th USENIX Conference on Hot Topics in Security*, HotSec'09, page 9, USA, 2009. USENIX Association.

[21] C. Katsini, Y. Abdrabou, G. E. Raptis, M. Khamis, and F. Alt. The Role of Eye Gaze in Security and Privacy Applications: Survey and Future HCI Research Directions. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–21, New York, NY, USA, 2020. ACM.

[22] Kaya and Bilge. Deep Metric Learning: A Survey. *Symmetry*, 11(9):1066, 2019.

[23] A. Kupin, B. Moeller, Y. Jiang, N. K. Banerjee, and S. Banerjee. Task-Driven Biometric Authentication of Users in Virtual Reality (VR) Environments. In I. Kompatsiaris, B. Huet, V. Mezaris, C. Gurrin, W.-H. Cheng, and S. Vrochidis, editors, *MultiMedia Modeling*, volume 11295 of *Lecture Notes in Computer Science*, pages 55–67. Springer International Publishing, Cham, 2019.

[24] 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, Sept. 2016.

[25] M. Levi, I. Hazan, N. Agmon, and S. Eden. Behavioral embedding for continuous user verification in global settings. *Computers & Security*, 119:102716, 2022.

[26] J. Liebers, M. Abdelaziz, L. Mecke, A. Saad, J. Auda, U. Gruenefeld, F. Alt, and S. Schneegass. Understanding User Identification in Virtual Reality Through Behavioral Biometrics and the Effect of Body Normalization. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 2021.

[27] J. Liebers, S. Brockel, U. Gruenefeld, and S. Schneegass. Identifying users by their hand tracking data in augmented and virtual reality. *International Journal of Human–Computer Interaction*, 0(0):1–16, 2022.

[28] J. Liebers, P. Horn, C. Burschik, U. Gruenefeld, and S. Schneegass. Using Gaze Behavior and Head Orientation for Implicit Identification in Virtual Reality.

In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21, pages 1–9, New York, NY, USA, 2021. Association for Computing Machinery.

[29] J. Liebers, P. Laskowski, F. Rademaker, L. Sabel, J. Hoppen, U. Gruenefeld, and S. Schneegass. Kinetic signatures: A systematic investigation of movement-based user identification in virtual reality. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[30] C. Löffler, L. Reeb, D. Dzibela, R. Marzilger, N. Witt, B. M. Eskofier, and C. Mutschler. Deep Siamese Metric Learning: A Highly Scalable Approach to Searching Unordered Sets of Trajectories. *ACM Transactions on Intelligent Systems and Technology*, 13(1):1–23, 2022.

[31] S. Marcel, M. S. Nixon, J. Fierrez, and N. Evans, editors. *Handbook of biometric anti-spoofing: presentation attack detection*. Advances in computer vision and pattern recognition. Springer, Cham, Switzerland, second edition edition, 2019.

[32] F. Mathis, J. Williamson, K. Vaniea, and M. Khamis. RubikAuth: Fast and Secure Authentication in Virtual Reality. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, pages 1–9, New York, NY, USA, 2020. Association for Computing Machinery.

[33] Mehdi. Image similarity estimation using a siamese network with a contrastive loss, 2023. Available online at https://keras.io/examples/vision/siamese_contrastive, last retrieved on July 15, 2024.

[34] R. Miller, N. K. Banerjee, and S. Banerjee. Using Siamese Neural Networks to Perform Cross-System Behavioral Authentication in Virtual Reality. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pages 140–149, Lisboa, Portugal, 2021. IEEE.

[35] S. Morgan. Tensorflow addons losses: TripletSemiHardLoss, 2023. Available online at https://www.tensorflow.org/addons/tutorials/losses_triplet, last retrieved on July 15, 2024.

[36] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No Fuss Distance Metric Learning using Proxies. Available online at http://arxiv.org/abs/1703.07464v3, last retrieved on July 15, 2024.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[38] K. Pfeuffer, M. J. Geiger, S. Prange, L. Mecke, D. Buschek, and F. Alt. Behavioural Biometrics in VR: Identifying People from Body Motion and Relations in Virtual Reality. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[39] P. Refaeilzadeh, L. Tang, and H. Liu. *Cross-Validation*, pages 532–538. Springer US, Boston, MA, 2009.

[40] A. Rosebrock. Anomaly detection with Keras, TensorFlow, and Deep Learning , 2023. Available online at https://pyimagesearch.com/2020/03/02/anomaly-detection-with-keras-tensorflow-and-deep-learning/, last retrieved on July 15, 2024.

[41] M. A. Sasse, M. Steves, K. Krol, and D. Chisnell. The Great Authentication Fatigue – And How to Overcome It. In *Cross-cultural design*, volume 8528 of *Lecture notes in computer science Information systems and application, incl. Internet/web and HCI*, pages 228–239. Springer, Cham, 2014.

[42] C. Schell, K. Kobs, T. Fernando, A. Hotho, and M. E. Latoschik. Extensible Motion-based Identification of XR Users using Non-Specific Motion Data. Available online at http://arxiv.org/abs/2302.07517v2, last retrieved on July 15, 2024.

[43] A. Schmidt. Implicit human computer interaction through context. *Personal Technologies*, 4(2-3):191–199, 2000.

[44] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[45] J. Serrà, S. Pascual, and A. Karatzoglou. Towards a universal neural network encoder for time series, 2018. https://arxiv.org/pdf/1805.03908.pdf, last retrieved: July 15, 2024.

[46] P. Tanisaro and G. Heidemann. Time series classification using time warping invariant echo state networks. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 831–836, Anaheim, CA, USA, 2016. IEEE.

[47] 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)*, pages 1578–1585, Anchorage, AK, USA, 2017. IEEE.

[48] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[49] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao. Time series classification using multi-channels deep convolutional neural networks. In F. Li, G. Li, S.-w. Hwang, B. Yao, and Z. Zhang, editors, *Web-Age Information Management*, pages 298–310, Cham, 2014. Springer International Publishing.