


# Drinking Task Pipeline

Diese Pipeline führt eine Klassifikation von Trinkbewegungen durch.

Die Elemente, welche angepasst werden können, um Ansätze auszuprobieren, wurden mit  gekennzeichnet.

Geeignete Parameter (Beispiel):

## 3.2 Ansatz auswählen

- Ansatz 1: Absolute Datenpunkte mit geschnittenen Videos
- `n_frames`: **38**

## 4.3 Modell anwenden

- `n_neurons`: **64**
- RNN-Modell

# 1. Import Dependencies

```
In [ ]: import pandas as pd
import numpy as np

from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

from tensorflow import keras
from tensorflow.keras import layers

from sklearn.model_selection import RandomizedSearchCV

import math
```

# 2. Landmarks extrahieren

```
In [ ]: def extract_landmarks_normal():
        %run extract_landmarks.py --video-dir data/video/normal --output-file

In [ ]: def extract_landmarks_compensation():
        %run extract_landmarks.py --video-dir data/video/compensation --output-file

In [ ]: # Falls die Landmarks neu extrahiert werden sollen, folgende Befehle ausk
# extract_landmarks_normal()
# extract_landmarks_compensation()
```

## 3. Data Pre-Processing

```
In [ ]: # Landmarks einlesen
data_normal_original = pd.read_csv('data/landmarks_normal.csv')
data_compensation_original = pd.read_csv('data/landmarks_compensation.csv')

# Augmentierte Datenpunkte für Datenanalyse einlesen
data_normal_augmented = pd.read_csv('../AvatarDataAugmentation/Data/Ke
data_compensation_augmented = pd.read_csv('../AvatarDataAugmentation/D
data_compensation_augmented2 = pd.read_csv('../AvatarDataAugmentation/

# Zusätzliche aus Unity generierte Datenpunkte einlesen
data_normal_augmented_unity = pd.read_csv('../AvatarDataAugmentation/D
data_compensation_augmented_unity = pd.read_csv('../AvatarDataAugmenta

data_normal = pd.concat([data_normal_original, data_normal_augmented, dat
data_compensation = pd.concat([data_compensation_original, data_compensat

# Label setzen (Kompensation = 1, keine Kompensation = 0)
data_compensation.compensation = 1
data_normal.compensation = 0

data = pd.concat([data_compensation, data_normal], axis=0)
data[data.frame == 1]
```

```
Out [ ]:
```

	path	frame	compensation	
0	data/video/compensation\01_trinken_kompensatio...	1	1	1000
239	data/video/compensation\02_trinken_kompensatio...	1	1	1000
529	data/video/compensation\03_trinken_kompensatio...	1	1	1020
828	data/video/compensation\04_trinken_kompensatio...	1	1	1020
1174	data/video/compensation\05_trinken_kompensatio...	1	1	1020
...	...	...	...	...
64581	/Users/salomekoller/Library/CloudStorage/OneDr...	1	0	1070
64787	/Users/salomekoller/Library/CloudStorage/OneDr...	1	0	1090
64994	/Users/salomekoller/Library/CloudStorage/OneDr...	1	0	1100
65199	/Users/salomekoller/Library/CloudStorage/OneDr...	1	0	1140
65405	/Users/salomekoller/Library/CloudStorage/OneDr...	1	0	1200

624 rows x 102 columns

### 3.1 Data Centering

```
In [ ]: """
Zentrierung Datenpunkte für Datenanalyse
Dies ist die Zentrierung des Avatars am Kopf, indem dieser zum Punkt (0,0,0)
Alle anderen Keypoints werden entsprechend diesen X, Y und Z Werten versch
```

Die Zentrierung wird anstelle des von Nils erstellten Ansatzes verwendet, damit die Daten für die Datenanalyse im späteren Schritt verwendet werden

```
"""
x_cols = []
y_cols = []
z_cols = []
for i in range(33):
    x_cols.append(f'x_{i}')
    y_cols.append(f'y_{i}')
    z_cols.append(f'z_{i}')
```

```
In [ ]: videos_raw = list(data.groupby(data.path))
len(videos_raw)
```

```
Out[ ]: 602
```

```
In [ ]: def center_keypoints(uncentered_videos):

    centred_df = pd.DataFrame()
    for path, video in uncentered_videos:
        keypoints = video[x_cols + y_cols + z_cols]

        # Step 2: get the head position and use it as center of mass
        center_of_mass = keypoints[['x_0', 'y_0', 'z_0']].iloc[0].values
        #print(center_of_mass)

        # Step 3: Translate keypoints
        tmp = pd.DataFrame()
        tmp[x_cols] = keypoints[x_cols] - center_of_mass[0]
        tmp[y_cols] = keypoints[y_cols] - center_of_mass[1]
        tmp[z_cols] = keypoints[z_cols] - center_of_mass[2]
        tmp.insert(0, 'path', path)
        tmp.insert(0, 'frame', video['frame'])
        tmp.insert(0, 'compensation', video['compensation'])

        centred_df = pd.concat([centred_df, tmp], axis=0, ignore_index=True)

    return centred_df
df_centred = center_keypoints(videos_raw)
```

```
In [ ]: df_centred.head(5)
```

```
Out[ ]:
```

	compensation	frame	path	x_0	y_0	z_0
0	1	1	/Users/salomekoller/Library/CloudStorage/OneDr...	0.0000	-10.0000	-10.0000
1	1	2	/Users/salomekoller/Library/CloudStorage/OneDr...	-3.3216	-10.0000	-10.0000
2	1	3	/Users/salomekoller/Library/CloudStorage/OneDr...	1.6550	-10.0000	-10.0000
3	1	4	/Users/salomekoller/Library/CloudStorage/OneDr...	3.9860	-10.0000	-10.0000
4	1	5	/Users/salomekoller/Library/CloudStorage/OneDr...	5.4320	-10.0000	-10.0000

5 rows x 102 columns

## 3.2 Export centered Data for Data Analysis

### 3.2.1 Export Datenset "Kompensiert"

```
In [ ]: df_centred_augmented = df_centred[df_centred.path.str.startswith('/Users/')]
df_centred_original = df_centred[df_centred.path.str.startswith('data/')]

df_centred_augmented_compensated = df_centred_augmented[df_centred_augmented.compensation == 1]
df_centred_augmented_compensated.head(5)
```

```
Out [ ]:
```

	compensation	frame	path	x_0	y_0
0	1	1	/Users/salomekoller/Library/CloudStorage/OneDr...	0.0000	-10.0000
1	1	2	/Users/salomekoller/Library/CloudStorage/OneDr...	-3.3216	-10.0000
2	1	3	/Users/salomekoller/Library/CloudStorage/OneDr...	1.6550	-10.0000
3	1	4	/Users/salomekoller/Library/CloudStorage/OneDr...	3.9860	-10.0000
4	1	5	/Users/salomekoller/Library/CloudStorage/OneDr...	5.4320	-10.0000

5 rows × 102 columns

```
In [ ]: df_centred_augmented_compensated.to_csv('../Data/KeypointsBereinigtKompensiert.csv',
                                                sep=";",
                                                encoding='utf-8',
                                                index=False)
```

### 3.2.2 Export Datenset "Nicht Kompensiert"

```
In [ ]: df_centred_augmented_notCompensated = df_centred_augmented[df_centred_augmented.compensation == 0]
df_centred_augmented_notCompensated.head(5)
```

```
Out [ ]:
```

	compensation	frame	path	x_0	y_0
1558	0	1	/Users/salomekoller/Library/CloudStorage/OneDr...	0.0000	-10.0000
1559	0	2	/Users/salomekoller/Library/CloudStorage/OneDr...	-3.3216	-10.0000
1560	0	3	/Users/salomekoller/Library/CloudStorage/OneDr...	1.6550	-10.0000
1561	0	4	/Users/salomekoller/Library/CloudStorage/OneDr...	3.9860	-10.0000
1562	0	5	/Users/salomekoller/Library/CloudStorage/OneDr...	5.4320	-10.0000

5 rows × 102 columns

```
In [ ]: df_centred_augmented_notCompensated.to_csv('../Data/KeypointsBereinigtNichtKompensiert.csv',
                                                    sep=";",
                                                    encoding='utf-8',
                                                    index=False)
```

```
In [ ]: df_centred_augmented = df_centred_augmented[['path', 'frame', 'compensation', 'x_12', 'y_12', 'z_12', 'x_13', 'y_13', 'z_13', 'x_14', 'y_14', 'z_14', 'x_15', 'y_15', 'z_15', 'x_16', 'y_16', 'z_16', 'x_17', 'y_17', 'z_17', 'x_18', 'y_18', 'z_18']]
df_centred_augmented.head(5)
```

```

        'x_20', 'y_20', 'z_20', 'x_21', 'y_21', 'z_21', 'x_22', 'y_22'
        'x_24', 'y_24', 'z_24', 'x_25', 'y_25', 'z_25', 'x_26', 'y_26'
        'x_28', 'y_28', 'z_28', 'x_31', 'y_31', 'z_31', 'x_32', 'y_32'

df_centred_original = df_centred_original[['path', 'frame', 'compensation
        'x_12', 'y_12', 'z_12', 'x_13', 'y_13', 'z_13', 'x_14', 'y_14'
        'x_16', 'y_16', 'z_16', 'x_17', 'y_17', 'z_17', 'x_18', 'y_18'
        'x_20', 'y_20', 'z_20', 'x_21', 'y_21', 'z_21', 'x_22', 'y_22'
        'x_24', 'y_24', 'z_24', 'x_25', 'y_25', 'z_25', 'x_26', 'y_26'
        'x_28', 'y_28', 'z_28', 'x_31', 'y_31', 'z_31', 'x_32', 'y_32']

```

```

In [ ]: # Videos Original gruppieren und in Liste schreiben
videos_raw_augmented = list(df_centred_augmented.groupby(df_centred_augme
len(videos_raw_augmented)

# Videos Augmentiert gruppieren und in Liste schreiben
videos_raw_original = list(df_centred_original.groupby(df_centred_origina
len(videos_raw_original)

```

Out[ ]: 599

### 3.3 Pre-Processing-Funktionen

```

In [ ]: '''
Diese Funktion bringt die rohen Videos in eine neue Form, ohne die Videos
Sie wird für den Sliding Window Ansatz benötigt.
Aus den rohen Videos werden nur noch die extrahierten Datenpunkte als List
'''

def remap_raw_videos(unmapped_videos):
    remapped_videos = []
    for video in unmapped_videos:
        v = video[1].reset_index()
        mapped_vid = v.loc[:, 'x_0':]
        remapped_videos.append(mapped_vid)
    return remapped_videos

```

```

In [ ]: '''
Schneiden der Videos

Diese Funktion bringt die rohen Videos in eine neue Form und schneidet si
vor und hinter dem vertikalen Höhepunkt der trinkenden Hand ab.

Aus den rohen Videos werden nur noch die extrahierten Datenpunkte als List

Inputs:
uncutted_videos: Die ungeschnittenen rohen Videos
n_frames: Die Anzahl der Frames, welche vor und nach dem Höchstpunkt der
'''

def cut_videos(uncutted_videos, n_frames=0):
    cut_videos = []
    for idx, video in enumerate(uncutted_videos):
        v = video[1].reset_index()
        # Position des höchsten Punktes (tiefster Wert, da tiefere Werte
        minpos = np.argmin(v.y_16)

        # Falls der Höhepunkt der Hand zu nahe am Beginn / Ende des Video
        # Das Video soll dann optimalerweise gelöscht werden.
        if((minpos < 20) or (len(v) - minpos < 20)):

```

```

        print('\033[91m' + 'Video mit dem Namen\n'
              + videos_raw[idx][0]
              + '\nund Index\n' + str(idx)
              + '\nzeigt keine korrekte Trinkbewegung. Bitte entferne

    cut_vid = v.loc[minpos-n_frames:minpos+n_frames, 'x_0':] # Vide
    cut_videos.append(cut_vid)
    return cut_videos

```

```

In [ ]: '''
Videos zentrieren

Diese Funktion zentriert alle Videos, in dem von allen Landmarks die Posi
'''
def center_data(uncentered_videos):
    centered_videos = []
    for video in uncentered_videos:
        centered_video = []
        # Position des Kopfes im ersten Frame des Videos bestimmen (x, y
        head_start = video.loc[:, 'x_0': 'z_0'].values[0]
        for frame in np.array(video):
            centered_frame = []
            # Frame reshappen, sodass alle Landmarks als eine Liste zählen
            landmarks = frame.reshape((21, 3))
            for landmark in landmarks:
                centered_frame.append(landmark - head_start)
            centered_video.append(list(np.array(centered_frame).flatten()))
        centered_videos.append(centered_video)
    return centered_videos

```

```

In [ ]: '''
Relative Abstände

Diese Funktion berechnet den Abstand jedes Punktes des Skeletts zum Kopf

Shape der Rückgabe: [x Anzahl Videos, x Anzahl Frames, 33 Datenpunkte]
'''
def calc_distances(raw_videos):
    # Abstand zu Kopf
    distances = []

    for video in raw_videos:
        frame_distances = []
        for frame in np.array(video):
            points = frame.reshape((33, 3))
            point_distances = []
            for k in range(len(points)):
                # Distanz einzeln von x-, y- und z-Koordinaten
                distance = points[k]-points[0]
                # Distanz mittels Formel berechnen
                point_distances.append(math.sqrt(distance[0]**2 + dista
            frame_distances.append(point_distances)
        distances.append(frame_distances)
    return distances

```

```

In [ ]: '''
Diese Funktion gibt die Labels der Videos zurück.
1 = Compensation
0 = Natural

```

```
'''
def define_labels():
    labels = []

    for i in range(len(videos_raw_original)):
        labels.append(np.mean(videos_raw_original[i][1].compensation))

    for i in range(len(videos_raw_augmented)):
        labels.append(np.mean(videos_raw_augmented[i][1].compensation))

    return labels
labels = define_labels()
```

```
In [ ]: '''
Sliding Windows

Diese Funktion erstellt für alle Videos von unslided_videos Sliding Windows.
Ausserdem werden die Labels auf die Sliding Windows korrekt verteilt.

Shape der Rückgabe: [x Anzahl Sliding Windows, {window_size} Anzahl Frame]
'''
def create_sliding_windows(unslided_videos, window_size):
    videos_slided = []
    unslided_labels = define_labels()
    slided_labels = []
    for idx, unslided_video in enumerate(unslided_videos):
        video_label = unslided_labels[idx]
        for i in range(len(unslided_video) - window_size + 1):
            videos_slided.append(unslided_video[i:i+window_size])
            slided_labels.append(video_label)
    return videos_slided, slided_labels
```

```
In [ ]: '''
In dieser Funktion werden die Daten in eine geeignete Form gebracht.
Es werden die Values der einzelnen Videos in eine Liste geschrieben und zurückgegeben.
'''
def reshape_videos(unshaped_videos):
    reshaped_videos = []
    labels = []
    for video in unshaped_videos:
        reshaped_videos.append(video.values)
    return reshaped_videos
```

### 3.4 Ansatz auswählen

```
In [ ]: # Ansatz 1: Absolute Datenpunkte mit geschnittenen Videos
# videos = reshape_videos(cut_videos(videos_raw, n_frames=38))

# Ansatz 2: Zentrierte Datenpunkte mit geschnittenen Videos
# videos = center_data(cut_videos(videos_raw, n_frames=10))

# Ansatz 3: Relative Datenpunkte (Abstände zum Kopf) mit geschnittenen Videos
# videos = calc_distances(cut_videos(videos_raw, n_frames=10))

# Ansatz 4a: Sliding Windows mit absoluten Datenpunkten
videos_original, labels_original = create_sliding_windows(center_data(reshape_videos(videos_raw), labels_raw))
videos_augmented, labels_augmented = create_sliding_windows(center_data(reshape_videos(videos_raw), labels_raw))
```

```
# Ansatz 4b: Sliding Windows mit zentrierten Datenpunkten
# videos, labels = create_sliding_windows(center_data(remap_raw_videos(vi

# Ansatz 4b: Sliding Windows mit relativen Datenpunkten
# videos, labels = create_sliding_windows(calc_distances(remap_raw_videos

np.array(videos_original).shape
np.array(videos_augmented).shape
```

Out[ ]: (5741, 10, 63)

```
In [ ]: '''
Falls es zu einem Fehler kommt, kann folgende Zeile angepasst und auskomm
Alternativ kann das Video im Ordner gelöscht werden und die Datenpunkte n
'''
# videos_raw.remove(videos_raw[144])
```

Out[ ]: '\nFalls es zu einem Fehler kommt, kann folgende Zeile angepasst und auskommentiert werden, um ein falsches Video zu entfernen.\nAlternativ kann das Video im Ordner gelöscht werden und die Datenpunkte neu extrahiert werden. =\n'

## 3.5 Features und Labels definieren

```
In [ ]: # Features bestimmen
X = np.asarray(videos_original)
X.shape
```

Out[ ]: (142748, 10, 63)

```
In [ ]: # Labels setzen
y = np.array(labels_original)

y.shape
```

Out[ ]: (142748,)

```
In [ ]: # Features bestimmen
X_train_salome = np.asarray(videos_augmented)
X_train_salome.shape
```

Out[ ]: (5741, 10, 63)

```
In [ ]: # Labels setzen
y_train_salome = np.array(labels_augmented)

y_train_salome.shape
```

Out[ ]: (5741,)

```
In [ ]: # Daten in ein Trainings- und Testset unterteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

```
In [ ]: """
Augmentierte Daten werden dem Trainingsset hinzugefügt
"""
# Concatenate the additional data with the existing X_train and y_train
```



```

X_train = np.concatenate((X_train, X_train_salome), axis=0)
y_train = np.concatenate((y_train, y_train_salome), axis=0)

# Ensure the shape of X_train and y_train matches after concatenation
assert X_train.shape[0] == y_train.shape[0]

# Optionally, shuffle the concatenated data
indices = np.arange(X_train.shape[0])
np.random.shuffle(indices)
X_train = X_train[indices]
y_train = y_train[indices]

```

```

In [ ]: # Daten in ein Trainings- und Testset unterteilen
X_validate, X_test, y_validate, y_test = train_test_split(X_test, y_test,

```

```

In [ ]: # Verteilung der Testdaten
np.mean(y_test)

```

```

Out[ ]: 0.5771260449259795

```

## 4. Modelle trainieren

### 4.1 LSTM

```

In [ ]: def define_lstm_model(input_shape, n_neurons=1):
        lstm_model = keras.Sequential()
        lstm_model.add(layers.LSTM(n_neurons, activation='relu', input_shape=
        lstm_model.add(layers.Dense(1, activation='sigmoid'))
        return lstm_model

```

### 4.2 RNN

```

In [ ]: def define_rnn_model(input_shape, n_neurons=1):
        rnn_model = keras.Sequential()
        rnn_model.add(layers.SimpleRNN(n_neurons, activation='relu', input_sh
        rnn_model.add(layers.Dense(1, activation='sigmoid'))
        return rnn_model

```

### 4.3 Modell anwenden

```

In [ ]: '''
        Gewünschtes Modell erstellen

        Auskommentieren und Anzahl Neuronen anpassen (n_neurons)
        input_shape wird automatisch übernommen
        '''

        # Anzahl Neuronen bestimmen 🖋️
        n_neurons=64

        # LSTM
        # model = define_lstm_model(n_neurons=n_neurons, input_shape=X.shape[1:])

```

```
# RNN
model = define_rnn_model(n_neurons=n_neurons, input_shape=X.shape[1:])
```

/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
super().\_\_init\_\_(\*\*kwargs)

```
In [ ]: # Modell kompilieren
model.compile(
    loss='binary_crossentropy',
    optimizer="Adam",
    metrics=['accuracy'],
)
```

```
In [ ]: X_train.shape
```

```
Out[ ]: (105664, 10, 63)
```

```
In [ ]: # Modell trainieren
model.fit(
    X_train, y_train, validation_data=(X_validate, y_validate), epochs=10
)
# validation_data=(X_validate, y_validate),
```

```
In [ ]: model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	
simple_rnn_1 (SimpleRNN)	(None, 64)	
dense_1 (Dense)	(None, 1)	

Total params: 24,773 (96.77 KB)

Trainable params: 8,257 (32.25 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 16,516 (64.52 KB)

## Modell speichern

```
In [ ]: # model_name = 'models/LSTM/lstm_a_cv_'
# model_name = 'models/RNN/rnn_a_cv_'
# model_name = 'models/sliding_window/sw_lstm_a_cv_'
model_name = 'models/sliding_window/sw_rnn_a_cv_'

# Wenn das Modell gespeichert werden soll, folgende Zeile auskommentieren
# LEGACY
# model.save(model_name + str(X.shape[1]) + 'f_' + str(n_neurons) + 'n/'

model.save(model_name + str(X.shape[1]) + 'f_' + str(n_neurons) + 'n/' +
```

## 4.4 Hyperparametertuning

```
In [ ]: '''
Hyperparameter tuning

Diese Funktion findet die Anzahl an Neuronen, welche das beste Resultat l
'''
def tune_hyperparameters(untuned_model):
    # Regressor mit dem mitgegebenen Modell erstellen
    keras_regressor = keras.wrappers.scikit_learn.KerasRegressor(untuned_

    # Eine Suche mit dem Regressor erstellen. Es werden Werte der Neurone
    randomized_search_cv = RandomizedSearchCV(
        keras_regressor,
        {"n_neurons": np.arange(1, 128)},
        n_iter=10,
        cv=3
    )

    randomized_search_cv.fit(X_train, y_train, validation_data=(X_validat

    print(randomized_search_cv.best_params_)

    return randomized_search_cv.best_estimator_.model
```

```
In [ ]: # Auskommentieren und Modell anpassen, um das beste Modell zu finden und
# best_model = tune_hyperparameters(untuned_model=define_lstm_model)
# best_model.save('models/best_model')
```

## 5. Predictions

```
In [ ]: # Auskommentieren und ein bestimmtes Modell zu laden
# model = keras.models.load_model('models/RNN/rnn_a_cv_21f_32n')
```

```
In [ ]: result = model.predict(X_test)

result
```

670/670 ————— 0s 506us/step

```
Out[ ]: array([[1.0000000e+00],
               [1.0000000e+00],
               [1.0000000e+00],
               ...,
               [2.1799360e-03],
               [1.0000000e+00],
               [3.2184005e-04]], dtype=float32)
```

## 6. Evaluation

```
In [ ]: scores = model.evaluate(X_train, y_train, verbose=0)
print("In-Sample Accuracy: %.2f%%" % (scores[1]*100))
scores = model.evaluate(X_test, y_test, verbose=0)
print("Out-of-Sample Accuracy: %.2f%%" % (scores[1]*100))
```


In-Sample Accuracy: 99.35%  
Out-of-Sample Accuracy: 99.24%

```
In [ ]: result = (result > 0.5).flatten()
        actual = y_test > 0.5
```

```
In [ ]: confusion_matrix(actual, result)
```

```
Out[ ]: array([[ 9001,    54],
               [   108, 12250]])
```

## Evaluation eines einzelnen Videos

```
In [ ]: # Gewünschtes Sliding-Window-Modell auskommentieren 

model = keras.models.load_model('models/sliding_window/sw_rnn_a_cv_5f_64n
# model = keras.models.load_model('models/sliding_window/sw_lstm_a_cv_5f_

# model = keras.layers.TFSMLayer('models/sliding_window/sw_rnn_a_cv_10f_6
# model = keras.models.load_model('models/sliding_window/sw_rnn_a_cv_77f_
# model = keras.models.load_model('models/sliding_window/sw_lstm_a_cv_10f
# model = keras.models.load_model('models/sliding_window/sw_rnn_a_cv_20f_
# model = keras.models.load_model('models/sliding_window/sw_lstm_a_cv_20f
```

```
In [ ]: '''
Video klassifizieren

Diese Funktion berechnet für ein Video die Kompensationswahrscheinlichkeit

Input:
video_index: Index des Videos
window_size: Grösse der Sliding Windows

Output: Liste mit Wahrscheinlichkeiten für alle Sliding Windows der Video
'''
def predict_video(video_index, window_size=5):
    return model.predict(np.array(create_sliding_windows(remap_raw_videos
```