Machine Learning Workshop

International Society of Biomechanics Conference

Fukuoka, 2023

Dr. Marion Mundt

marion.mundt@uwa.edu.au

Theoretical Background

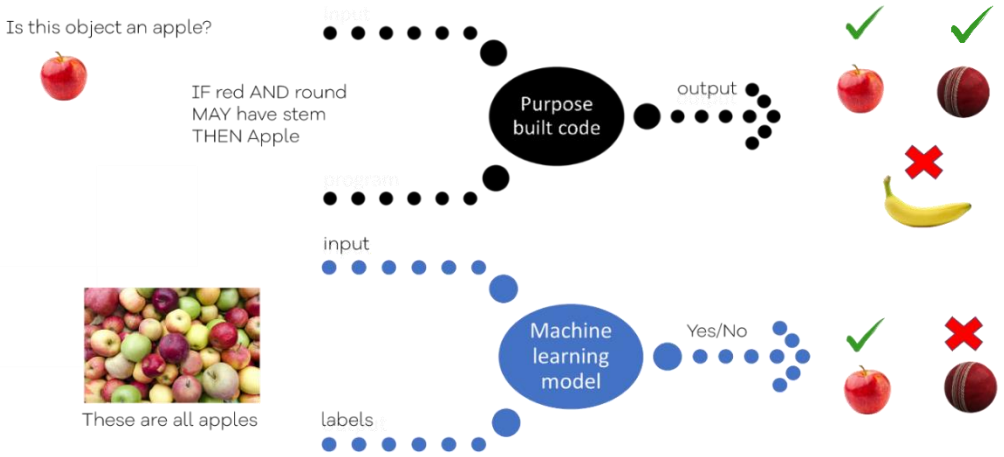
In the following sections, basic principles of machine learning will be introduced. You will learn about:

1. A brief introduction to machine learning
2. Machine learning tasks
3. Machine learning types
4. Artificial neural networks
5. Training and cross-validation
6. Machine learning benefits

The information provided is based on:
Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning.
https://www.deeplearningbook.org/

A brief introduction to machine learning

Machine learning has been defined as: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* (Mitchell, 1997).



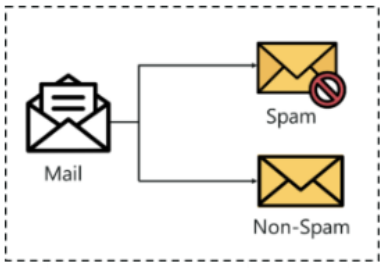
The image above represents the classification of fruit. A purpose-built algorithm will use if-statements to describe an apple such as: *if the object is red and round, the object is an apple*. This condition will detect that a banana is not an apple (it is neither red nor round), but a cricket ball will be confused as an apple. A machine learning model will learn patterns itself by seeing a lot of different items (inputs) that are either labelled as *apples* or *no apples* and adapt to be able to differentiate.

To link this example back to the definition by Mitchell, this means, that in contrast to a purpose-built code, that relies on conditional statements (if-statements), machine learning models learn from *experiences* (large datasets) and the evaluation of their *performance* (correct/incorrect) for the specific *task* (detecting apples). This occurs using statistical and mathematical equations that adapt to describe the relationship between input and output data.

Machine learning tasks

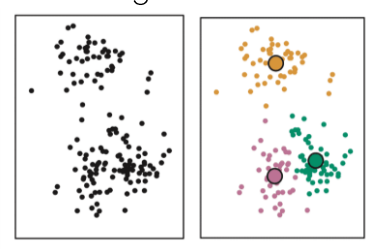
The three main machine learning tasks – classification, clustering, and regression – are shown below. For all different tasks, the aim is that the model learns connections based on provided ground truth data to apply the trained model for the estimation on unknown examples.

Classification



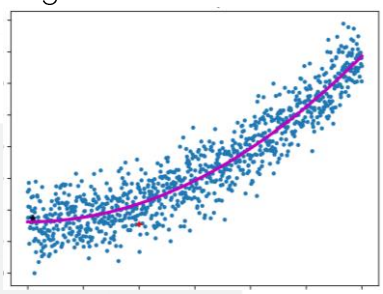
The input and target value are known, the output is a categorical, singular value.

Clustering



The input is grouped in several specified or unspecified categories, based on commonalities. The target is an unknown categorical, singular value.

Regression

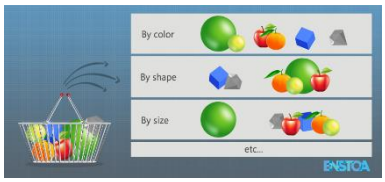


The input and target value are known, the output is a continuous, exact value.

An example for a classification task will be presented in this workshop: we will classify a group of people into two sub-groups (binary classification) based on given labels of 'healthy' or 'impaired' gait patterns. An example for a clustering task is the detection of similarities in the movement strategy during changes of direction to automatically find sub-groups. A well-known biomechanics example for a regression task is the estimation of the ground reaction force of a person walking based on their kinematics.

Machine learning types

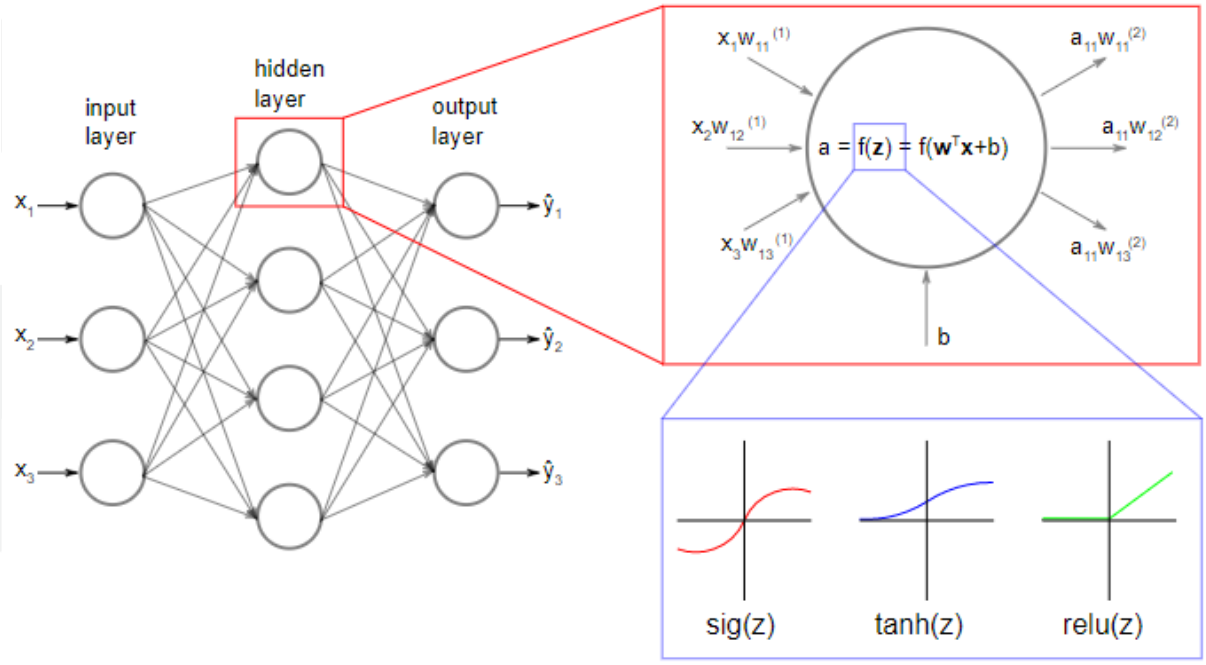
There are three main types of machine learning – supervised, unsupervised, and reinforcement learning. The differences between the three are displayed below.

Supervised Learning	Unsupervised Learning	Reinforcement Learning
		
Target values are used to train the machine learning model.	The machine learning model detects and learns hidden structures in input data. .	The input and output is unknown, and the machine learning model learns correct associations and actions based on a rewards system.

In this workshop, we will use artificial neural networks as a supervised machine learning method, because the labels to our gait data are known and can be provided to the algorithm. To use unsupervised learning, a clustering algorithm could be used to determine structures in the data and assign different sub-groups. Reinforcement learning can perceive and interpret its environment, take actions, and learn through trial and error. It has recently been used in biomechanics to model movement control mechanisms using biomechanical models.

Artificial neural networks

An artificial neural network is a type of machine learning model. It can be described as a series of connected mathematical operations and functions that are self-adjusted to best associate input data with output data.



Each artificial neural network is made up of at least three layers of nodes as displayed in the image above. The **input layer** is used to feed the data into the network and must have the same size as the inputs. In the **hidden layers**, the weights and biases are added (red box), activation functions (blue box) are utilised, and the model is optimised. The number and size of hidden layers can vary and is responsible for the complexity of the model. The more and the larger the hidden layers, the higher the complexity and computational cost of the model. The **output layer** produces the results of the neural network and must be the same size as the ground truth output. Each node within each layer is representative of the activation function. Typically, the activation function used for a particular node will be the same for all other nodes in the same layer. The connections between each of the nodes is representative of the weight multiplication and bias addition.

There are many different structures of networks, that are designed for different use-cases. In this workshop, we will focus on the simplest architecture: dense neural networks.

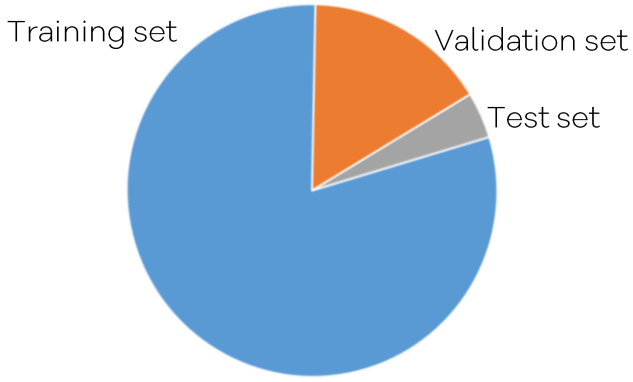
In general, training artificial neural networks involves four major steps:

1. Multiplication of input data with randomly assigned weights (w) and the addition of a bias (b) term to the weighted input data. Weights and bias are the learnable parameters of an artificial neural network. While the weights control how strong the influence of a neuron is on the results, the bias term guarantees that even with all inputs being zero, an activation of the following layer occurs.
2. Calculation of output data using activation functions at each node. Weights and bias are introduced to the output of all subsequent layers (except the final output layer).
3. Final network output using current set of weights and bias is compared to the ground truth output.
4. The weights are adjusted to predict outputs best associated to ground truth data through a process known as back-propagation. This process is conducted whilst aiming to minimise a specified loss function. The loss is a strong indicator on the estimation

accuracy of the model and can be used to detect underfitting (the model is not adapting well to the training data) or overfitting (the model is not generalising to new test data) of the model.

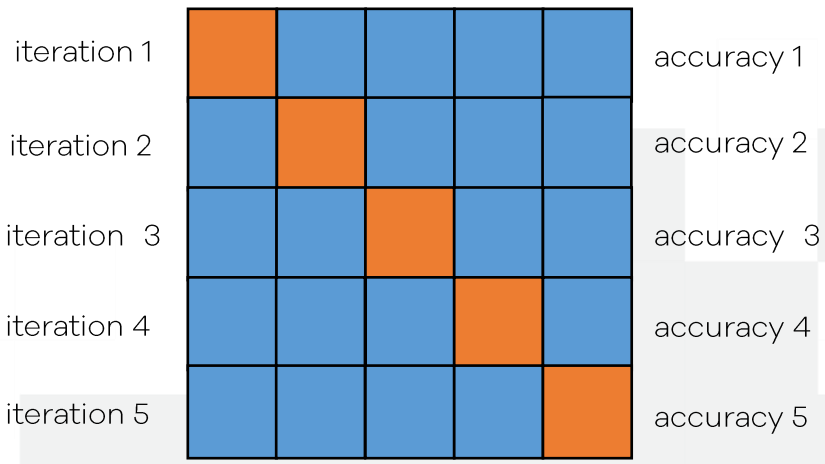
Training and cross-validation

Artificial neural networks are trained and validated on one dataset that contains all necessary input and output data for the given task. After training, they are tested on former unseen data to evaluate generalisability. For this process, input and ground truth output data needs to be present. Finally, the model can be used for data with unknown outputs.



To train a neural network (or machine learning algorithm in general), the dataset needs to be split into three different subsets: the first and largest subset contains about 80% of the data and is used for training. The second set is used for the evaluation of the model: 80% of this dataset, the validation set, is used to assess the performance during training, while the third and smallest set, the test set, is used to test the performance of the model after training. The suggested split of the dataset is just a rule of thumb and can differ dependent on the overall dataset.

To ensure the highest quality and best generalisability of results, cross-validation is recommended. This is especially important for small datasets as those usually found in biomechanics. Cross-validation is performed by splitting the dataset into different subgroups of training/validation/test data multiple times.



In biomechanics, a leave-one-subject-out cross-validation, as displayed above, is recommended. For this purpose, data of one participant is held back as a test set to ensure that the model has not seen any data of this particular participant during training. This process should ensure that the model generalises to new data.

For each iteration, the accuracy of the model is determined. The final accuracy of the model is stated as the average over all test sets. Regularly differences in accuracy between single participants can be observed.

Machine learning benefits

Machine learning offers quite a few benefits compared with purpose-built algorithms. Some of the most relevant in biomechanics are:

1. The ease of handling and batch-processing large datasets in an automated and significantly smaller amount of time than manual methods.
2. Detection of hidden patterns within datasets that may be too large or too complicated for the eye to see.
3. The trained model can be used to estimate unknown outputs from similar but unseen input data.

Learning outcomes

In this Theory section, basic principles of machine learning have been introduced. You have learned about:

1. A brief introduction to machine learning
2. Machine learning tasks – classification, clustering, regression
3. Machine learning types – supervised, unsupervised, reinforcement learning
4. Artificial neural networks – layers, nodes, weights, bias, activation functions
5. Training and cross-validation – dataset splits, leave-one-subject-out validation
6. Machine learning benefits

For further reading:
Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning.
https://www.deeplearningbook.org/

Preliminaries

In this section you will find the necessary information to:

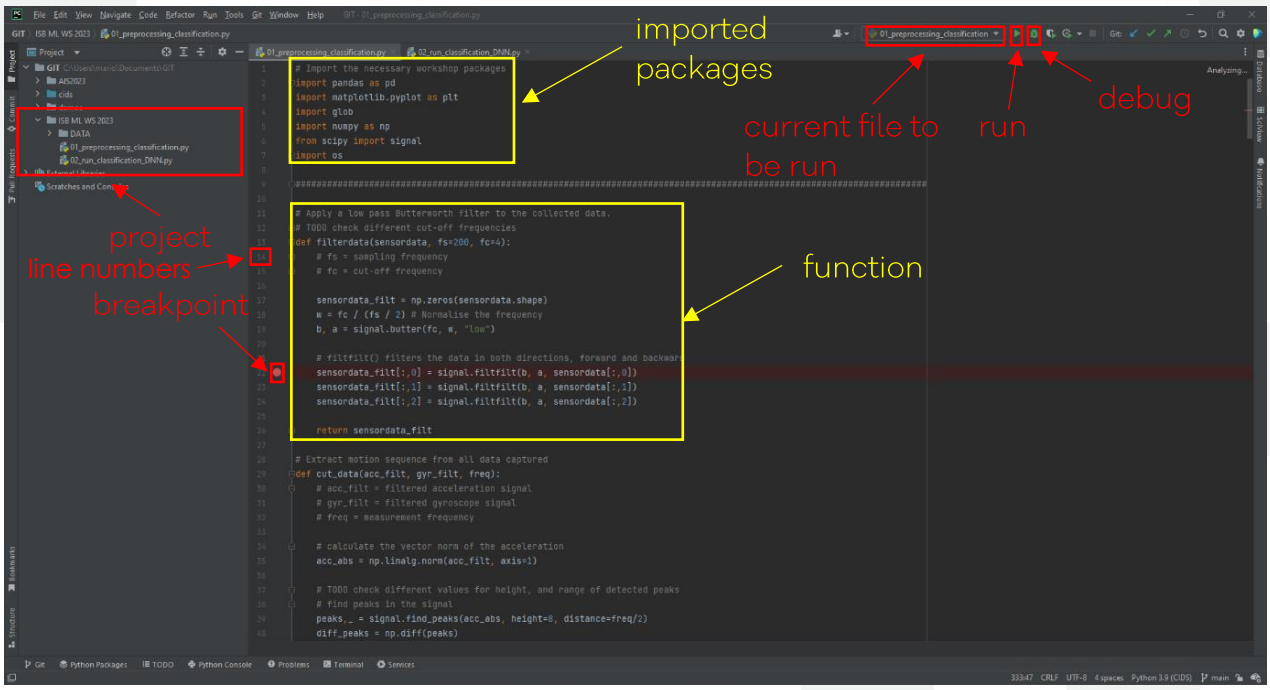
1. Access the workshop using a remote desktop connection.
2. Navigate in PyCharm, the python editor used in this workshop.

General overview of the PyCharm Editor

This workshop will be using PyCharm as Python editor. The figure and table below display the most relevant features of the editor.

The current project and its files can be found on the left-hand side. On top, the current file that will be executed is displayed with the run and debug button on the right. To debug a script, a breakpoint must be set by clicking in the relevant line. The script will only be executed up to that point and all variables will be displayed.

On top of the script, the imported packages that are used can be found. These packages contain predefined functions that can be used. Custom functions that are used in the script are defined below. By pressing ctrl and clicking on functions used in the script, you will be taken to the line where the function is defined.



Section	Usage
project	current project and its files
breakpoint	left click in line to set breakpoint to debug file
line numbers	line numbers will be used as reference throughout the workshop
current file to be run	file that will be executed when pressing run/debug

run	execute file
debug	execute file until breakpoint and show all variables
imported packages	imported packages that contain predefined functions that are used within the script
function	custom functions that have been defined; pressing ctrl and clicking on functions used in the script takes you to the line where the function is defined

Python and Python packages

For this workshop, we will use a virtual machine to ensure easy setup and everyone working with the exact same setup. We will use the following Python version and packages:

Anaconda virtual environment with Python version 3.11.

Packages

absl-py	1.4.0	pip	23.1.2
astunparse	1.6.3	protobuf	4.23.3
bzip2	1.0.8	pyasn1	0.5.0
ca-certificates	2023.05.30	pyasn1-modules	0.3.0
cachetools	5.3.1	pyparsing	3.1.0
certifi	2023.5.7	python	3.11.3
charset-normalizer	3.1.0	python-dateutil	2.8.2
contourpy	1.1.0	pytz	2023.3
cycler	0.11.0	requests	2.31.0
flatbuffers	23.5.26	requests-oauthlib	1.3.1
fonttools	4.40.0	rsa	4.9
gast	0.4.0	scipy	1.10.1
google-auth	2.20.0	seaborn	0.12.2
google-auth-oauthlib	1.0.0	setuptools	67.8.0
google-pasta	0.2.0	six	1.16.0
grpcio	1.54.2	sqlite	3.41.2
h5py	3.8.0	tensorboard	2.12.3
idna	3.4	tensorboard-data-server	0.7.1
jax	0.4.12	tensorflow	2.12.0
keras	2.12.0	tensorflow-estimator	2.12.0
kiwisolver	1.4.4	tensorflow-intel	2.12.0
libclang	16.0.0	tensorflow-io-gcs-filesystem	0.31.0
libffi	3.4.4	termcolor	2.3.0
markdown	3.4.3	tk	8.6.12
markupsafe	2.1.3	typing-extensions	4.6.3
matplotlib	3.7.1	tzdata	2023.3
ml-dtypes	0.2.0	urllib3	1.26.16
numpy	1.23.5	vc	14.2
oauthlib	3.2.2	vs2015_runtime	14.27.29016
opencv-python	4.8.0.74	werkzeug	2.3.6

openssl	3.0.8	wheel	0.38.4
opt-einsum	3.3.0	wrapt	1.14.1
packaging	23.1	xz	5.4.2
pandas	2.0.2	zlib	1.2.13
pillow	9.5.0		

Part 1 – Classification of Gait

The aim of this first part of the workshop is to classify gait into *normal* and *impaired* based on inertial sensor data collected with smartphones. There are a few group assignments to test (a) different thresholds when cleaning the data, and (b) the influence of different input parameters on the classification accuracy. The following section will step through this process. You will learn about:

1. The dataset used.
2. Steps to pre-process the provided data.
3. Training an artificial neural network using TensorFlow.
4. Presentation of results.

Dataset

The dataset used in this part of the workshop was collected in Germany in 2019 and contains data of 21 participants. Each participant gave informed written consent to participate.

Each participant performed three walking trials, each lasting for about 30s containing straight walking and turns. After three normal walking trials, a small box was placed into the participant's right shoe to cause an impaired walking pattern. With this impairment, the participants performed another three walking trials of about 30s. Inertial sensor data (acceleration and angular rate) was collected with two different smartphones placed in the participants back pockets using the open-source application Phyphox.

Data pre-processing

Hands on: A01_preprocessing_classification.py

Go to line 130! Set breakpoints next to line numbers in the script and debug through the following steps to familiarise yourself with the data and code.

To enable/disable plots set variable *plot* to True/False (ll. 130-131).

Data extraction – extract data from files and perform first processing steps.

1. Load data of each file in a loop (ll. 138-145)
2. Extract acceleration signal, angular rate signal, and frequency (ll. 147-157).
3. Filter data using a Butterworth filter (ll. 159-161). The necessary function is defined on top (ll. 12-25)
4. Determine motion sequence based on peaks in the acceleration signal and cut trial accordingly (ll. 189-190). The necessary function is defined on top (ll. 27-50).
5. Perform principal component analysis/singular value decomposition to align data to account for different sensor alignments in different smartphones (e. g. x-axis of IMU in one smartphone might point towards the screen, while pointing towards the top in another phone) (ll. 200-203). The necessary functions are defined on top (ll. 52-91).

Data formatting – ensuring data is formatted in a suitable way to be handled and manipulated.

1. Cut data to similar sequences (strides, steps) based on y-axis angular rate. There are multiple solutions for this, using different peak detection parameters (ll. 237-239).
2. Store each sequence of 3D acceleration and 3D angular rate in a list (ll. 249-251).
3. Resample each sequence to 100% to normalise different sequence lengths (ll. 253-255).
4. Store all time-normalised sequences in a 3D array that can be used as input to the neural network later (ll. 257-259), concatenate the acceleration and angular rate signal (ll. 261-262), and save the files (ll. 264-266).

Summary – after performing the previous steps, all data is stored in *numpy* arrays that contain single, similar motion sequences. The information of which class an array belongs to is stored in the filename; the data is divided into the targeted classes *normal* and *impaired*.

To prepare input and output data for the classification, further steps are necessary.

Cleaning – Remove inconsistent data or outliers in the dataset that can be a result of errors due to data measurement equipment or human error during data collection. Since the data was collected in an uncontrolled setting with different devices, the previously described pre-processing steps are not perfect and might lead to the detection of inconsistent motion sequences. This could be avoided using the same device for data collection and a more controlled data collection setting. However, this would not represent a simple measurement setup. A device-specific pre-processing (especially peak detection) could be applied, or further cleaning steps as outlined in the following section.

1. Load and combine all input data, create output data (0 for normal, 1 for impaired gait), and an array containing subject number (ll. 271-305).
2. Divide data based on their class (ll. 307-314).
3. Add subject number to input array to enable a leave-one-subject-out cross-validation (ll. 316-320).
4. Clean sequence based on RMSE threshold (ll. 322-327). Perform this step with the threshold according to your group assignment so we can compare the classification results based on different input data.
    a. Group 1: threshold 6
    b. Group 2: threshold 8
    c. Group 3: threshold 10
    d. Group 4: threshold 12
    e. Group 5: threshold 14

    The necessary function is defined on top (ll. 93-122).
5. Store cleaned input data (ll. 325-331).

Time to switch to the next script! Hands on: A02_run_classification.py

1. Load input and output data (ll. 9-12).

Flatten inputs – dense neural networks cannot handle multidimensional time series data as such. To account for this, all inputs need to be flattened (concatenated to single vector).

1. Flatten input data (ll. 14-29). In this step, choose the input data based on your group assignment so we can compare the classification results we get based on different input data.

   a. Group 1: input type "both"
   b. Group 2: input type "acceleration"
   c. Group 3: input type "angular rate"
2. Extract output and subject number from trial (ll. 31-33).

Cross-validation – based on the extracted subject numbers a leave-one-subject-out validation can be performed. This can easily be done using a for-loop.

1. Define training, validation and test subjects without data of one participant being part of more than one subset (ll. 37-56).
2. Exclude test data if it contains less than 10 trials (ll. 58-60).
3. Normalise all inputs to mean of zero and standard deviation of one using mean and standard deviation of the training dataset (ll. 62-68).
4. Check if dataset is balanced (contains roughly as many positive samples as negative ones) (ll. 70-73).

Artificial Neural Network – in this workshop a simple dense neural network is defined as displayed in the table below. Not all parameters have been discussed in this workshop, hence further reading information has been provided.

Input layer	Size based on input type (606 for both, 303 otherwise)
Hidden layer	Size 128
Output layer	Size 1
Regularisation	l1 = 0, l2 = 0.1, dropout = 0.4 For more info see: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036

Activation function	Input and hidden layer: relu; output layer sigmoid (necessary for binary classification)
Optimiser	Adam (learning rate = 0.0001) For more info see: https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d
Loss function	Binary cross entropy For more info see: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
Batch size; epochs	64; 50 For more info see: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

1. Define, compile, and train neural network model (ll. 75-88).
2. Test and evaluate the trained model using confusion matrices for each test subject (ll. 90-99).
3. Determine overall accuracy, precision, and recall (ll. 101-122).

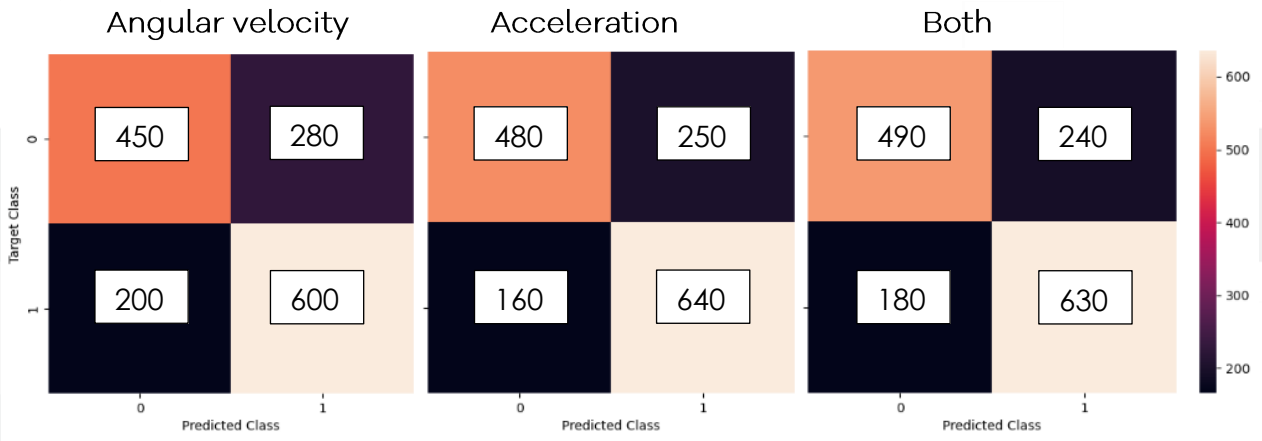
Summary – after performing the previous steps, an artificial neural network model has been trained to classify walking data into *normal* and *impaired* gait patterns. The model has been tested using a leave-one-subject-out cross-validation. To evaluate results, confusion matrices have been plotted and overall accuracy, precision, and recall been calculated.

Results

This section will summarise the most relevant results of the classification. The results presented are based on the group assignments, showing accuracy, precision, and recall for different cleaning thresholds (threshold 6, 1539 samples and threshold 14, 3540 samples) and input parameters (angular rate, acceleration, combination of both).

The performance of a classification model can be analysed easiest using confusion matrices. In these matrices, the number of true positives (top-left), false negatives (top-right), false positives (bottom-left), and true negatives (bottom-right) are displayed. True values (positive or negative) show correct predictions, while false values (positive or negative) show erroneous predictions.

Confusion matrices for threshold 6



Different metrics can be used to analyse the performance of a classification based on the confusion matrix:

Accuracy – standard metric, but not necessarily the one-and-only model metric:

$$accuracy = \frac{true\ negatives + true\ positives}{true\ negatives + false\ negatives + \ true\ positive + false\ positives}$$

Precision – good measure when the costs of False Positive (normal classified as impaired) is high:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

Recall – good measure when the costs of False Negative (impaired is classified as normal) is high:

$$recall = \frac{true\ positives}{true\ positive + false\ negatives}$$

	Accuracy		Precision		Recall	
	THR 6	THR 14	THR 6	THR 14	THR 6	THR 14
Angular velocity	0.686	0.644	0.616	0.637	0.692	0.628
Acceleration	0.728	0.693	0.652	0.682	0.745	0.681
Both	0.730	0.679	0.674	0.691	0.738	0.659

Summary – the results show that a dataset with higher quality samples leads to a higher estimation accuracy, precision and recall than the larger, low-quality dataset. The use of all six input parameters leads to better results than the use of only angular velocity or acceleration.

In this workshop only a few options to improve the performance of a machine learning model have been explored. Different filter, peak-detection thresholds and data cleaning steps could be explored. Additionally, not only the input data influences the results, also the choice of algorithm and architecture of model and should be investigated.

Part 2 – Pose estimation

The aim of this second part of this workshop is to understand and work with pose estimation keypoints. We will not run any pose estimation as there are several open-access models such as OpenPose or AlphaPose available, that are ready to be used. Instead, we will focus on post-processing steps needed to use pose estimation outputs.

For two examples of different pose estimation models, please refer to:

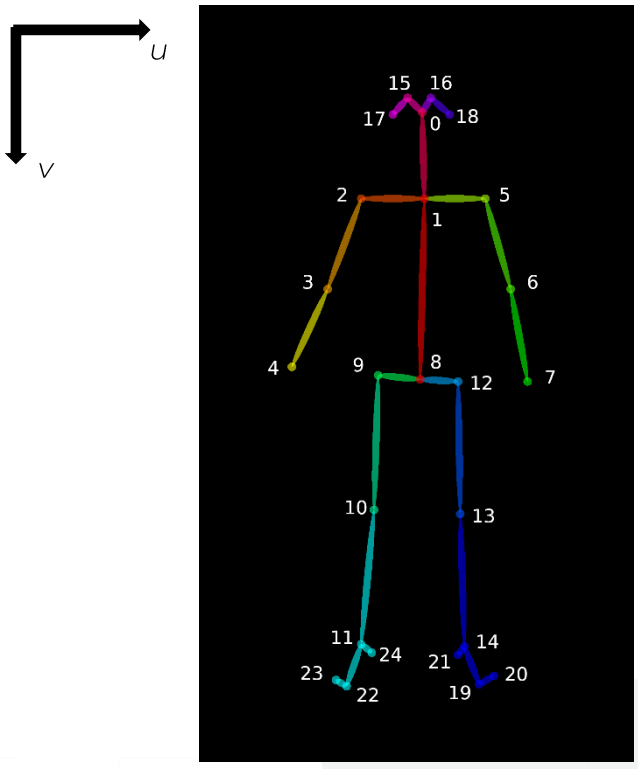
https://github.com/CMU-Perceptual-Computing-Lab/openpose
https://github.com/MVIG-SJTU/AlphaPose

In this part of the workshop, you will learn to:

1. Analyse the outputs of OpenPose.
2. Post-process the outputs to correct errors.
3. Visualise the postprocessed outputs.

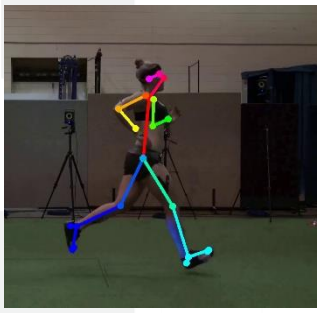
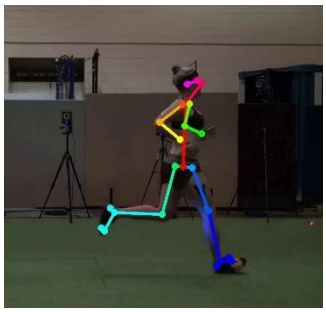
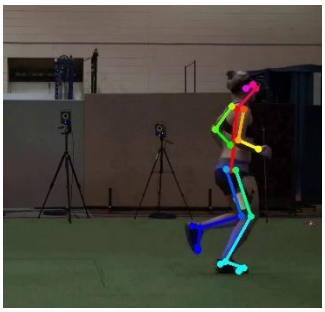
Dataset

The data used in this part of the workshop contains a single video .mp4 file that has been processed using the OpenPose Body_25 model. In each frame of the video, anatomically related keypoints as displayed below have been estimated.

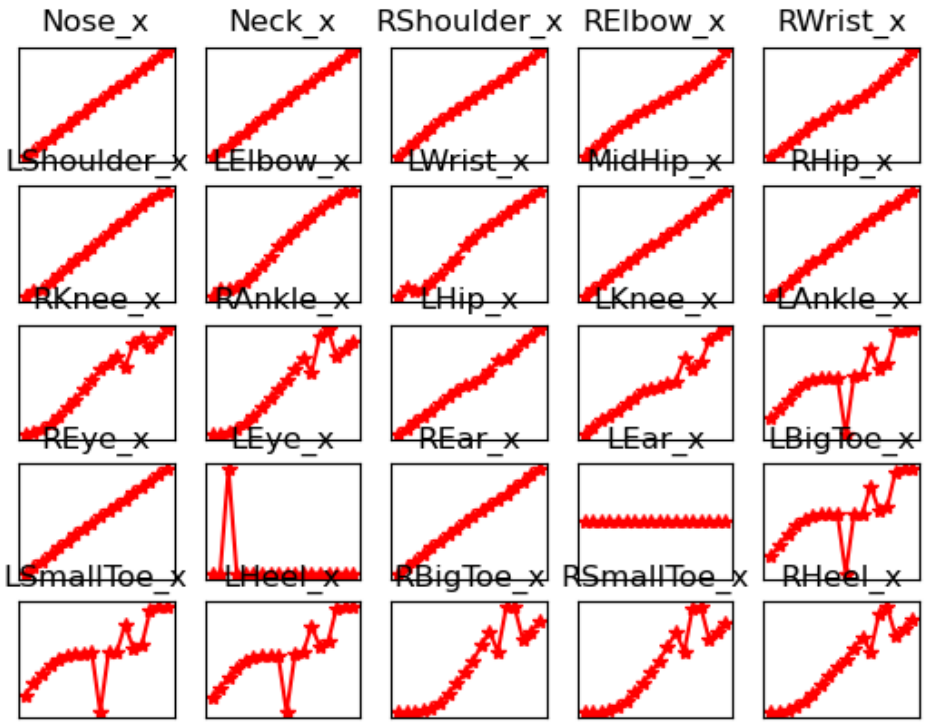


The model outputs a .json file for each frame containing the positional information of each keypoint in *u, v*-coordinates. The .json files have been concatenated to a single .csv file for further processing.

OpenPose outputs

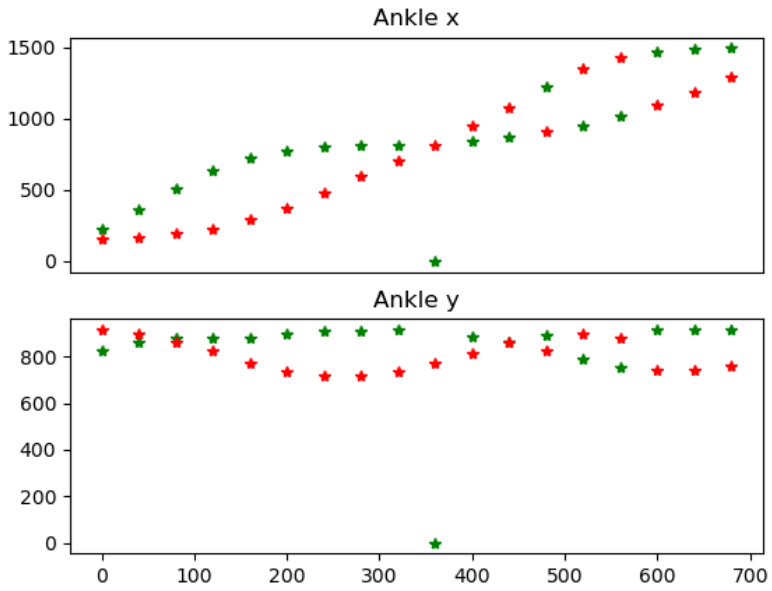
In the above sequence of images, we can see that the left and right leg have been swapped between frames. This is a common problem when using 2D pose estimation from a sagittal camera view and needs to be fixed in post-processing steps.



When plotting the keypoint trajectories, the jumps in keypoints when swapping left and right become visible. Further, we can see a dropout of keypoints in single frames on the left leg when this keypoints returns a zero, and keypoints that have not been detected (left eye and left ear) that also return zero.

In the following hands-on part, post-processing methods will be presented that help improve the pose estimation quality.

Post-processing of keypoints



The above figure shows the x and y component of the ankle trajectory. The right side is displayed in red, the left side in green. There are different errors that can be observed: There is a dropout in the left trajectory just after 350ms, the keypoint value is zero. After around 400ms, the left and right side is swapped in some frames.

Hands on: B01_filter_interpolate_OpenPose.py

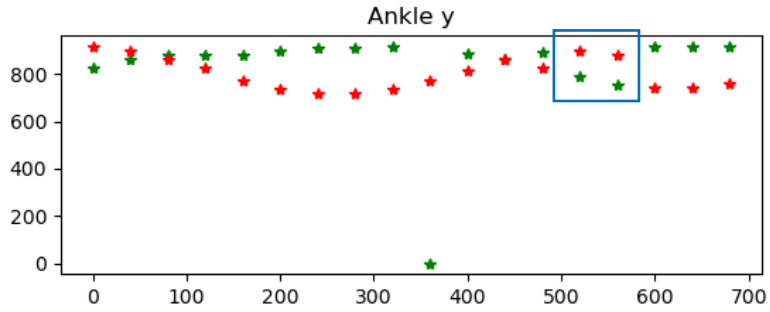
Go to line 347! Set breakpoints in the script and debug through the following steps to familiarise yourself with the data and code.

Load data – extract data from file and perform first processing step.

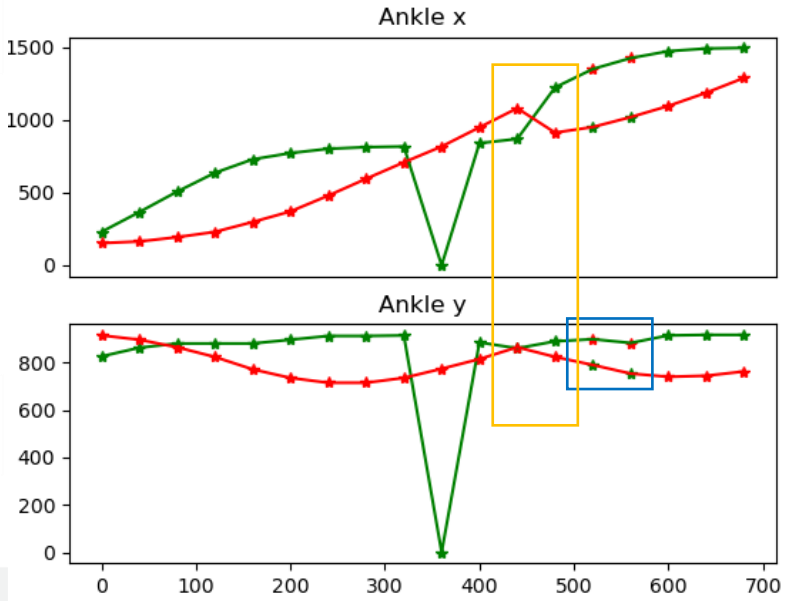
1. Load keypoints from .csv file (ll. 347-349).
2. Replace nan values (frames where no keypoint has been detected) by zeros for further processing (ll. 350).
3. Cut the trial to relevant frames that completely show the participant. This can be based on the foot contact in this example (ll. 352-356).
4. Create a copy of the original keypoints to ensure that this variable is not overwritten in the following steps (ll. 358-359).

Conditional filter – in the following step a conditional filter is used to correct keypoints that swapped sides and interpolate keypoints that dropped out.

1. A custom function has been created based on different conditions to correct erroneous data (ll. 361-362). For each step a copy of the data frame is created to avoid overwriting.
   Reminder: ctrl + click on the function name takes you to its definition!

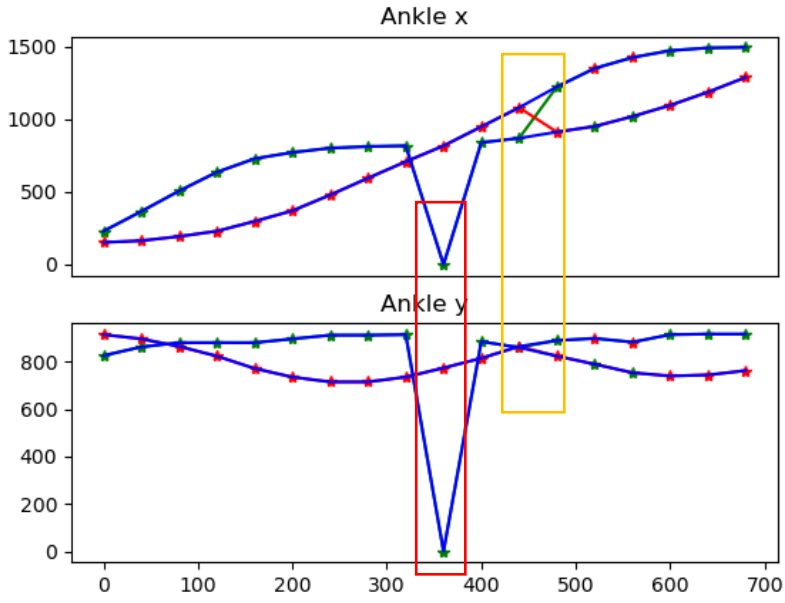


a. Find frames where the y-coordinate of the ankle keypoint jumps a predefined threshold. In this case, a threshold of 70px is chosen to detect the frames displayed in the blue box (ll. 27-36).
b. Swap left and right of the lower-body keypoints based on the detected indices (ll. 38-67).

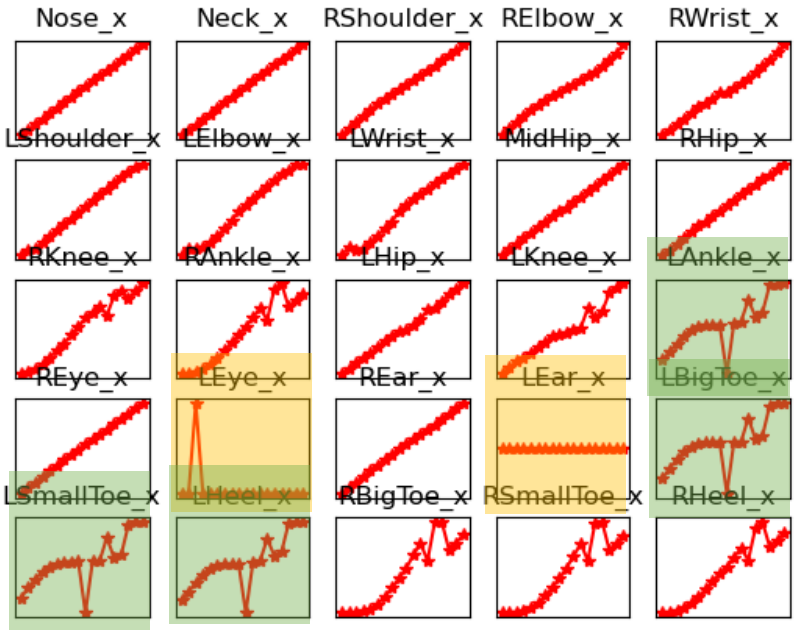


The solid lines in the above figure show, that flipped keypoints were successfully detected and changed, but the left and right side is still swapped after 400ms. This cannot be detected based on a threshold in the y-coordinate, as they are very similar when the legs cross, but in the x-coordinate.

c. Find frames where the x-coordinate of the ankle keypoint jumps a predefined threshold. In this case, a threshold of 150px is chosen to detect the frame displayed in the yellow box (ll. 69-78).
d. Swap left and right of the lower-body keypoints based on the detected index (ll. 80-109).

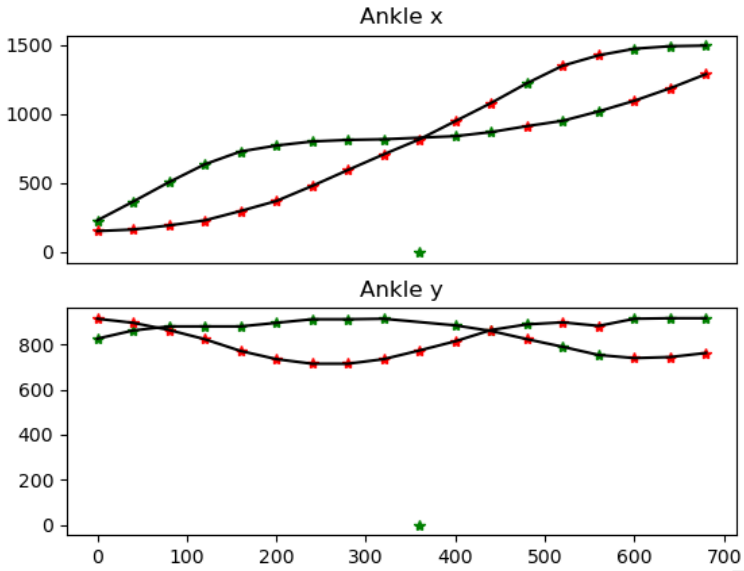


The solid blue lines in the above figure show, that swapped sides were successfully detected and changed. The last error in the data is the dropped out keypoint displayed in the red box.



Keypoints of the left foot (green boxes) dropped out for a single frame, while the left eye and left ear (yellow boxes) have only been detected in one or no frame at all.

e. Drop keypoints that are missing for more than 10 frames. 10 frames have been chosen as this is more than 50% of the trial (ll. 115-117).
f. Detect and interpolate keypoints that are missing in a single frame (ll. 119-122). This trial only shows gaps of a single frame, in other cases larger gaps might be interpolated similarly.

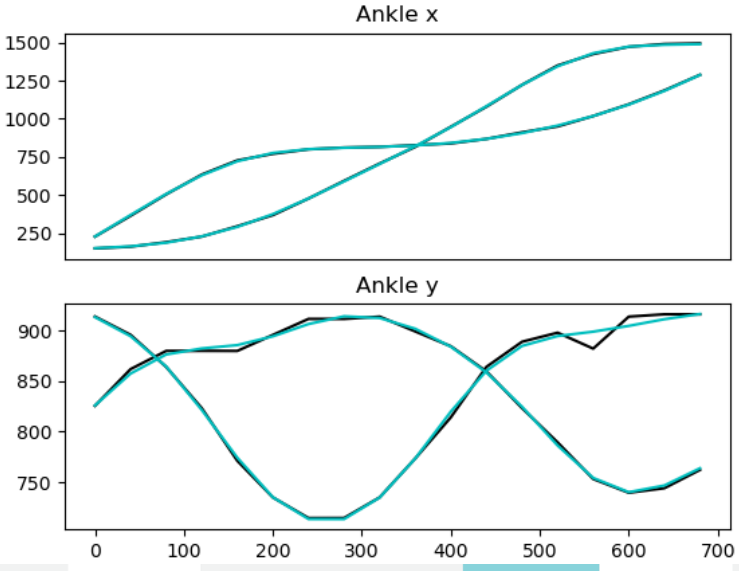


The solid black lines in the above figure show that erroneous keypoints have been successfully corrected.

Summary – in the above section keypoints that flipped the left and right side between frames have been detected and corrected. Looking at only the y-component of the ankle keypoint proved not to be sufficient to detect a consistent swap in left and right. By looking for jumps in the x-component of the ankle keypoint, this could be detected and fixed. Finally, keypoints that were not detected in a sufficient number of frames have been removed, while those missing in single frames only have been interpolated.

Butterworth filter – a Butterworth filter is applied to smooth the data as it is common practice in biomechanics.

1. A custom function has been created to apply a Butterworth filter to each keypoint (ll. 364-365).
    a. A 4th order Butterworth filter with a cut-off frequency of 10Hz has been designed and applied (ll. 17-23). The unfiltered keypoint trajectories are displayed in black, the filtered trajectories in cyan in the figure below.



Visualisation – simple line plots have been used to visualise every step of the processing steps above (ll. 367-373).
To better visualise the results, the processed keypoints can be overlayed on the original video (l. 376) or the original pose estimation video (l. 377). A custom function is used to create the animation (ll. 379-386).
The animation function (ll. 300-344) overlays the keypoints specified frame by frame on the video. Since the left eye and ear keypoint have not been detected in the original video, face keypoints have been excluded from the animation (ll. 306-309).

Summary – visualisation of data is necessary to explore potential errors in pose estimation. Simple line plots can be used to analyse single steps of a post-processing pipeline, while the final overlay of the video can be used to show the results in a similar way as OpenPose.

In the second part of this workshop strategies to analyse the outputs of OpenPose have been explored and custom post-processing steps to correct errors have been presented. These steps can only be seen as an example and are not necessarily applicable to a different dataset. Thresholds might differ based on camera resolution, frame rate, and type of movement.