Bianca Oswald, BSc

AI-assisted MiniLab for Gait Analysis on Quadrupeds

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn., Jörg Schröttner

Dr. Christoph Leitner

Institute of Health Care Engineering

Graz, November 2022

Abstract

AI-assisted MiniLab for Gait Analysis on Quadrupeds

Machine learning is already widely used in healthcare to aid physicians in the process of making diagnoses. The developed software applies such techniques to classify the gait of a dog as lame or healthy based on measured pressure data and gait analysis videos. For the body part detection on the videos, an open-source Convolutional Neural Network (CNN) named DeepLabCut (DLC) is used, providing a markerless pose estimation. Thanks to transfer learning, the final model achieved a cross-entropy loss of 0.00636 as well as a train error of 4.76 px and a test error of 5.45 px, even though only a small data set of 1160 frames from ten different patients was employed. The trained XGBoost classifier achieved an accuracy of 95.24% on previously unseen data, i.e., the test data, which was extracted from the training data set and not part of the training. However, the application of this model to new (unknown) test patients yielded only 58.33% accuracy, indicating that the model does not generalize to new subjects. To obtain a well-generalized model, a large amount of curated data is usually required, which means that the data set used for the training of the XGBoost classifier needs to be extended. The importance of data diversity as well as the feature selection is demonstrated by the performed parameter study.

Keywords: machine learning, markerless pose estimation, DeepLabCut, XGBoost classification, gait analysis on quadrupeds

Kurzfassung

AI-unterstütztes MiniLab für die Ganganalyse bei Vierbeinern

Maschinelles Lernen wird im Gesundheitswesen bereits häufig eingesetzt, um Ärzten bei der Erstellung von Diagnosen zu helfen. Die entwickelte Software nutzt solche Techniken, um den Gang eines Hundes anhand von gemessenen Druckdaten und Ganganalysevideos als lahm oder gesund zu klassifizieren. Für die Erkennung der Körperteile in Videos wird ein quelloffenes "neuronales Faltungsnetzwerk" namens DeepLabCut verwendet, das eine markerlose Posenschätzung ermöglicht. Dank des Transferlernens erreichte das endgültige Modell einen Kreuzentropieverlust von 0,00636 sowie einen Trainingsfehler von 4,76 px und einen Testfehler von 5,45 px, obwohl nur ein kleiner Datensatz von 1160 Bildern von zehn unterschiedlichen Patienten verwendet wurde. Der trainierte XGBoost-Klassifikator erreichte eine Genauigkeit von 95,24% bei zuvor ungesehenen Daten, d.h. den Testdaten, die aus dem Trainingsdatensatz extrahiert wurden und nicht Teil des Trainings waren. Die Anwendung dieses Modells auf neue (unbekannte) Testpatienten ergab jedoch nur eine Genauigkeit von 58,33%, was darauf hindeutet, dass das Modell nicht auf neue Probanden generalisiert werden kann. Um ein gut verallgemeinertes Modell zu erhalten, ist in der Regel eine große Menge an kuratierten Daten erforderlich, was bedeutet, dass der für das Training des XGBoost-Klassifikators verwendete Datensatz erweitert werden muss. Die Bedeutung der Datenvielfalt und der Merkmalsauswahl wird anahnd der durchgeführten Parameterstudie gezeigt.

Keywords: Maschinelles Lernen, Markerlose Posenschätzung, DeepLabCut, XGBoost Klassifikation, Ganganalyse bei Vierbeinern

Contents

Abbreviations

ML	machine learning
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DLC	DeepLabCut
ANN	Artificial Neural Networks
ReLU	rectified linear unit
ELU	exponential linear unit
SELU	scaled exponential linear unit
ResNet	Residual Network
GD	gradient descent
SGD	stochastic gradient descent
PA	Pressure Analyzer
RF	Random Forest
OOB	out-of-bag
VetMed	University of Veterinary Medicine Vienna
GUI	graphical user interface
CrLa	craniolateral
CrMe	craniomedial
CaLa	caudolateral
CaMe	caudomedial
FL	front left
FR	front right
HL	hind left
HR	hind right
MAE	mean average Euclidean error
RMSE	root mean square error
FFT	fast Fourier transform
ROC	receiver operating characteristics
AUC	area under curve

TPR	true positive rate
FPR	false positive rate

Terminology

Parameter	Learnable variables of an algorithm
Weights	Parameter in fully connected layers
Kernel	Parameter in convolutional layers
Hyperparameter	Variables determined by the researcher prior to the training process; no learnable variables
Epoch	One forward and one backward pass of all training examples
Batch size	Number of frames processed simultaneously
Iterations	Number of passes, each pass using [batch size] number of examples
Features; Columns	Synonym for attribute-value vectors representing input data for classifiers

1. Introduction

1.1. Gait analysis for dogs

This section briefly describes the topographical anatomy of dogs, their gait patterns, and different types of lameness with their associated symptoms. In addition, the Pressure Analyzer (PA), a state-of-the-art ground reaction force data acquisition tool that aids in the detection of lameness [1], is discussed.

1.1.1. Topographical anatomy, gait patterns, and lameness

To simplify orientation on the body as well as the body surfaces, a dog's body is split into lines and planes (see Figure 1.1). The dorsal (a) and ventral (b) midlines in the longitudinal direction define the body's center. They encompass the median surface (A), which divides the dog's body into the two halves left and right. The sagittal planes (B) run parallel to this median surface, subdividing the body into unequal areas. The transverse planes (C) are perpendicular to the longitudinal planes and refer to the transverse cross sections of the body. Finally, there are the dorsal planes (D), which are parallel to the dorsal body and perpendicular to the longitudinal and transverse planes. These planes are also called bilateral planes because symmetrical body sides appear in this view. [2]

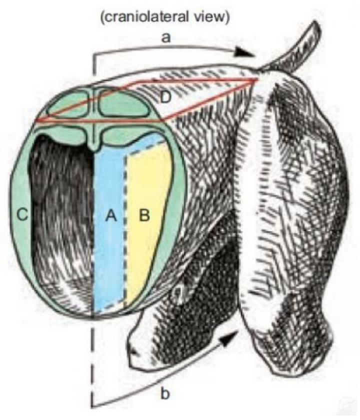


Figure 1.1: Topographical anatomy of a dog: The body's right and left halves are separated by the median plane (A), which connects the dorsal midline (a) and ventral midline (b). The sagittal planes (B) split the body longitudinally into unequal sections. While the transverse planes (C) subdivide the body in transverse direction, the dorsal planes (D) divide the body perpendicular to the longitudinal and transverse planes. Adapted from Figure 1. [2]

The phrases listed in the following table are used to describe the orientation and topographical relations of organs:

Term	Direction/relation	Body part
Caudal	Towards the tail	Body
Cranial	Towards the head	Body
Rostral	Towards the tip of the nose	Head
Dorsal	Related to the back or dorsum	Body
Ventral	Towards the belly	Body
Proximal	Towards the attached end	Axis of the body
Distal	Towards the free end	Axis of the body
Palmar	Surface of the manus (hand), faces caudally	Limb in normal standing attitude
Plantar	Surface of the pes (foot), faces caudally	Limb in normal standing attitude
Dorsal	From the carpus/tarsus distally	Thoracic/pelvic limb
Abaxial	Away from the axis	Central axis of the manus or pes
Axial	Towards the axis	Central axis of the manus or pes

Table 1.1: List of terms used in anatomy to describe directions and topographical relations of organs of dogs. [2]

Gait patterns

A thorough assessment of the dog's gait can detect gait patterns and, as a result, diagnose any lameness that may be present, greatly enhancing the identification of the dog's injured area [3]. The dog's gait can be split into symmetrical gaits such as the walk, trot, and pace and asymmetrical gaits such as the gallop [4]. The walk and trot are most commonly used in gait analysis, since the walk is the slowest and easiest gait to observe and the trot is the only gait where neither the fore nor the hind limb is supported by the contralateral[1] limb in bearing weight [5]. The locomotion begins with the front legs falling and the hind limbs propelling the dog forward at the same time. The forelimbs recover and the body's pitching movement is stabilized. The forward drive of the dog is mainly maintained by the propulsion of the hind limbs and forelimbs, whereby the center of gravity fluctuations of the dog are not as pronounced as in humans, for example. [3]

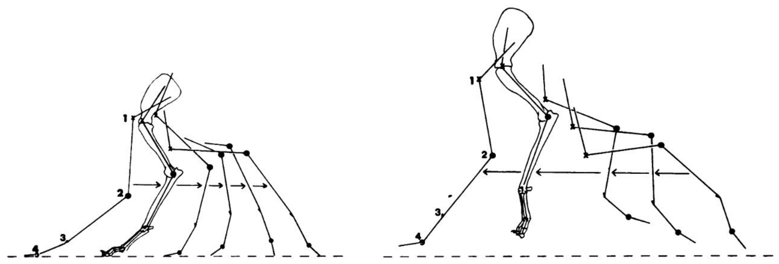


Figure 1.2: Forelimb movement during walking: The left image shows the stance phase of the step and the right image the swing phase. The direction of the movement is indicated in the figure by means of arrows and the joints are numbered as follows: 1-shoulder, 2-elbow, 3-carpal, and 4-metacarpophalangeal. Adapted from Figure 2. [3]

One entire cycle of leg movement, or step cycle, consists of the stance phase and the swing phase and is referred to as the stride. The initial half of the stance phase, which is caused by surface

[1] Contralateral comes from Latin and means "located on the opposite side or half of the body".

contact-induced braking forces, is followed by a propulsion phase, the second part of the stance phase. Throughout the entire stance phase, the paw remains in contact with the ground. The swing phase represents the time while the paw is in the air and consists of three parts: The propulsive force causes the leg to first swing backward; the muscles next cause it to swing forward for the movement; and finally, the muscles cause the leg to swing back towards the ground. During both the stance and swing phases, various forces act on the bone structure and the muscles of the legs. [5] Figure 1.2 shows these two phases for a forelimb movement during walking. As can be seen in the left picture of the figure, the leg is always touching the ground during the stance phase, while the leg is in the air during the swing phase (right picture). The joints are marked with numbers – 1-shoulder, 2-elbow, 3-carpal, and 4-metacarpophalangeal – and the arrows indicate the corresponding direction of the movement. [3]

Although all limbs are utilized to accelerate and decelerate, their functions are very different. Since the hind limbs of a dog essentially serve to generate the forward momentum of the body, this propulsive force arises earlier in their stance phase than in the stance phase of the forelimbs. Therefore, the hind limbs are more loaded during acceleration than the front limbs. The much more medial placement of the hind legs in the moment of contact has an additional positive effect on the exertion of this driven force. The front limbs on the other hand are basically used for restraint, with the main propulsive force required to move the front limbs being provided by the well-muscled shoulder region, which also serves to absorb the resulting shocks. Furthermore, the front limbs sustain around 60% of a dog's static weight. [3]

Walk and Trot

The walk is a symmetrical gait where the left and right legs complete the same movements but half a stride later. At any given time, the dog is supported by two, three or even four legs. Long-legged dogs keep their ipsilateral[2] leg pairs close together to avoid inferences between their limbs. As a result, only dogs with relatively long legs can pace. [3] While walking, the hind legs support the forward swing while the front legs assist to slow down and absorb shock. Although the weight is carried by all four limbs, the weight load on the front legs is greater. [4] Basically, the forelimb's stance phase is noticeably longer than the hind limbs'. However, after the swing phase of the fore legs is shorter than that of the hind legs,the stride cycle is still the same for all legs. Both the stance and the swing phases decrease as walking speed increases. [6] The length of the leg affects how long the paw stays in contact with the ground, so a dog with long legs has a longer contact time than a dog with short legs. The impact of both front feet at the beginning of each support phase is connected to a downward movement of the head and neck, whereas an upward movement is connected to the recovery of each limb. Compared to the forelimb, the hind limb is supported by a rather rigid structure thanks to the pelvis's connection with the sacrum. The lateral motions of the spinal column allow for horizontal movements of the hip joints in reference to the body. As a result, the dog moves its tail towards the hind limb, which is touching the ground (stance phase). [4]

The trot, is also a symmetrical gait, but with a little more speed than the walk. The dog is supported by alternating pairs of diagonal limbs, which normally touch the ground nearly at the same moment. However, some dogs go through a hovering phase that occurs in between the carrying stages of the gait. This phenomenon is known as the so called "flying trot". To ensure that the ground is cleared for the placement of the ipsilateral hind limb, the front limbs ground contact is substantially shorter than that of the hind limbs. [4] Long-legged dogs are more likely to encounter obstacles during the trot, so they rotate their bodies at an angle to the direction of movement, allowing the hind limb to pass beyond the forelimb without being obstructed. This activity is known as "crabbing", and is also present in the walk ("crab walking"). [3]

[2] Ipsilateral comes from Latin and means "located on the same side or half of the body".

Lameness

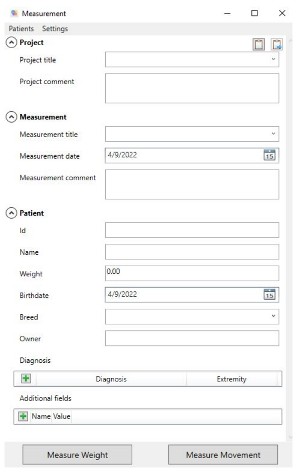
When a dog is lame, one or more limbs cannot support the dog optimally, resulting in additional weight bearing on the healthy limbs [4]. It is possible to detect such a shift by observing the dog's gait from a lateral, cranial, and caudal perspective and concentrating on the fluidity and coordination of the gait, the movement and position of the head and tail, asymmetries between the left and right legs as well as leg dropping and back arching [3].

Head nodding is a typical sign of **forelimb lameness**. The dog tends to raise its head when the lame forelimb touches the ground to lessen the weight on this leg. conversely, the head is lowered ("nodded") when the sound leg is supporting the dog's weight. [7] Moreover, the duration of the stance phase for the injured limb is reduced, while it is extended for the healthy limb. In addition to the longer stance phase, the sound leg also exhibits a greater stride length than the lame leg. [4] This aberrant behaviour of the front legs is often compensated by a more advanced posture of the hind limbs. The latter takes a greater role in supporting the dog, further reducing the weight that is passed through the lame front leg. [3]

Unlike the head action observable for forelimb lameness, dogs with **hind limb lameness** tend to nod their heads down when the injured hind leg touches the ground. This movement occurs because the dog is trying to shift its weight to the forelimbs, using its head and neck as a counterweight. [8] To further unload the lame leg, the dog increases the stride length of the two front limbs and shifts their extension during the stance phase further caudally [3]. Another sign of hind limb lameness is evident in the behavior of the tail. Healthy dogs swing their tails from one side to the other, while injured dogs move their tail up and down. The upward movement occurs, when the lame hind leg contacts the floor, further reducing the weight on the injured leg. [4] While lameness is more severe, and especially when moving faster, the animal may also have a "bunny hopping" gait in which both hind limbs are extended at once [8].

1.1.2. The Pressure Analyzer (PA)

The PA is a data acquisition tool (Pressure Analyzer 1.3.0.2; Michael Schwanda) used by the University of Veterinary Medicine Vienna (VetMed) to support the handling and visualization of data acquired with a pressure plate. It consists of two programs, the PA measurement tool used during the measurement process and the PA tool itself, where the results after the measurement are displayed. The Zebris FDM Type 2 pressure plate (Zebris Medical GmbH, Allgäu, Germany) used for the measurements contains 7040 sensors with a sampling rate of 100 Hz, is mounted in the middle of a 7 m walkway, and is covered with a rubber mat to avoid distraction or slippage. [1]



Project

Project title	
Project comment	

Measurement

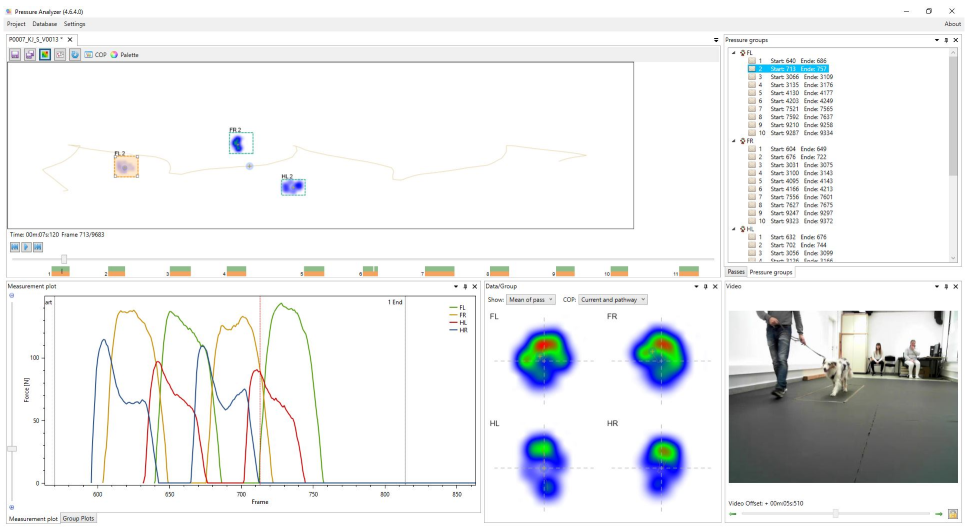
Measurement title	
Measurement date	4/9/2022
Measurement comment	

Patient

Id	
Name	
Weight	0.00
Birthdate	4/9/2022
Breed	
Owner	
Diagnosis	
Additional fields	

Figure 1.3: GUI of the PA measurement tool

Before the measurement starts, some basic information, such as patient ID and name, is provided in the PA measurement tool (see Figure 1.3). In the settings section, among other things, the make of the pressure plate and camera used are specified and the option to save the recorded video in a separate file can be selected. In addition, the "Measure Weight" button at the bottom left of the graphical user interface (GUI) determines the weight of the dog via the pressure plate. After all entries and settings have been made, the measurement can be started. [1]



```
FL
  1   Start: 640   Ende: 686
  2   Start: 713   Ende: 757
  3   Start: 3066  Ende: 3109
  4   Start: 3135  Ende: 3176
  5   Start: 4130  Ende: 4177
  6   Start: 4203  Ende: 4249
  7   Start: 7521  Ende: 7565
  8   Start: 7592  Ende: 7637
  9   Start: 9210  Ende: 9258
  10  Start: 9287  Ende: 9334
FR
  1   Start: 604   Ende: 649
  2   Start: 676   Ende: 722
  3   Start: 3031  Ende: 3075
  4   Start: 3100  Ende: 3143
  5   Start: 4095  Ende: 4143
  6   Start: 4166  Ende: 4213
  7   Start: 7556  Ende: 7601
  8   Start: 7627  Ende: 7675
  9   Start: 9247  Ende: 9297
  10  Start: 9323  Ende: 9372
HL
  1   Start: 632   Ende: 676
  2   Start: 702   Ende: 744
  3   Start: 3056  Ende: 3099
```

Figure 1.4: GUI of the PA tool: Visualization of the measurement data synchronously with the video. "FL" stands for the left front leg, "FR" for the right front leg, "HL" for the left hind leg, and "HR" for the right hind leg.

The results of the measurements are displayed via the PA tool, which consists of different areas. Figure 1.4 shows an example of a patient's data, where "FL" stands for front left, "FR" for front right, "HL" for hind left, and "HR" for hind right paws. In the lower right corner, the recorded video can be played back directly in the tool and the visible diagrams and measurement points are displayed synchronously with the video. [1]

Since the dog also moves outside the pressure plate during the gait analysis, there are sections in the video without measurement data and thus without diagrams. To easily and quickly display the parts with data, the PA tool includes the two tabs "Passes" and "Pressure groups", which can be seen in the upper right corner of Figure 1.4. On the first tab, all passes are listed where one pass corresponds to the time period in which the dog passes over the pressure plate exactly once. Since the corresponding start and end frames are also linked here, it is possible to jump directly to this part of the video by clicking on the listed pass. This basic idea of "jumping" is also possible within the second tab, which contains the individual pressure groups and distinguishes between the individual steps. [1]

For example, in Figure 1.4, ten steps are listed for the pressure group area of the front left (FL) paw, which means that the PA collected data of ten measurements for this paw throughout the examination. The collected data is associated with the corresponding step, including the frame number where the step began and ended. Thus, the marked pressure group in Figure 1.4 contains the data for the second step of the left front leg, which starts with frame 713 and ends with frame 757.

When one of the passes or a pressure group is selected, the rectangle symbolizing the pressure plate displays the corresponding paw prints. This rectangle may be found in the top left area of Figure 1.4, which also shows that marking the second step of the FL paw marks the corresponding paw print in the rectangle as well. Directly below it are the control buttons for the video and a line corresponding to the duration of the video. The green and orange boxes below this line symbolize the passes during the video, so it is very easy to see which pass is currently displayed. [1]

In the "Data/Group" area (in Figure 1.4 to the left of the video), the measured pressure for each stance phase of the paws is displayed in color. It is possible to select different representations, such as the mean value of all stance phases or only the last one. In the lower left corner, Figure 1.4 shows the graph area with the two tabs "Measurement plot" and "Group plots". The first tab depicts the force curves for each paw as a function of the frames in the video, while the second one shows the force curves as a function of the stance phase. Here, it is also possible to view the "Stance quadrant statistics", which includes the force curves for the craniolateral, craniomedial, caudolateral, and caudomedial areas of each paw. In addition, the PA tool provides a function for creating a report where all the data is exported to an Excel file template. Parameters that can be evaluated with the help of the PA tool include for example the peak vertical force, vertical impulse, stance phase duration, and the step length. [1]

1.2. Deep learning for markerless pose estimation

This section briefly describes the motivation for using machine learning (ML) techniques in motion analysis, the basic structure of a Deep Neural Network (DNN), and in particular CNNs. Additionally, the advantages of using residual networks and transfer learning are mentioned and an overview of common deep learning tools used for animal pose estimation is given.

1.2.1. Deep Neural Networks

Computerized tracking often involves placing reflective markers on animals, which can be very disruptive, and the quantity and position of these markers must be specified in advance [9]. Therefore, the currently used motion capture systems are very expensive and require highly skilled personnel, yet quantitative motion analysis is essential for decision making in medicine [10].

Thanks to new deep learning methods, powerful tools have been developed in recent years to support this type of measurement [11]. With this development, it has become possible to extract the posture of individuals with a standard video filmed with a simple smartphone, for example, instead of using laboratories and reflectors [10]. The ability to capture the movements of patients in their natural environment is especially crucial in the field of gait analysis [12], as it is possible that the patient will behave differently in the laboratory [10]. In addition, the usability of standard videos means a more cost-effective and easy-to-use application [10].

DNN algorithms used for pose estimation typically fall into the category of object detection [13], which not only identifies item types but also estimates the position of each object using a bounding box [14]. Prior to the era of deep learning, the object detection pipeline was split into three parts. In the first part, sliding windows were used to search for potential object locations in the picture, also known as regions of interest. From each of these sliding windows, a fixed-length feature vector was extracted in the second step to obtain semantic information about the detected area. In the last part, the region classifiers were taught how to categorize the regions covered. The fact that each stage of the detection pipeline was constructed and tuned independently prevented optimization of the overall system and was one of the causes of such detectors' limitations. [14]

However, object recognition algorithms based on DNNs can be optimized end-to-end and thus are able to attain accuracy comparable to that of humans [14]. The design of such systems was inspired by the human visual system, whose cells are controlled by feature detectors. In other words, when recognizing a human face, for example, each cell in the primary visual cortex processes only simple features like lines, curves, or dots, while the target components such as eyes or eyebrows are recognized in the higher-level visual cortex. The integration of more complex visual features with the context (e.g., the name) occurs in the posterior inferior temporal cortex, followed by the anterior inferior temporal cortex, which assigns its name to a specific context, e.g., is the person female or male. The evolved Artificial Neural Networks (ANN)s (which in recent years have been further developed into CNNs) mimic this visual system, i.e., the hierarchical structure, the ability for highly parallel processing, and the fact that the human system selects only the most relevant neurons along the spatial dimension within each layer. [15]

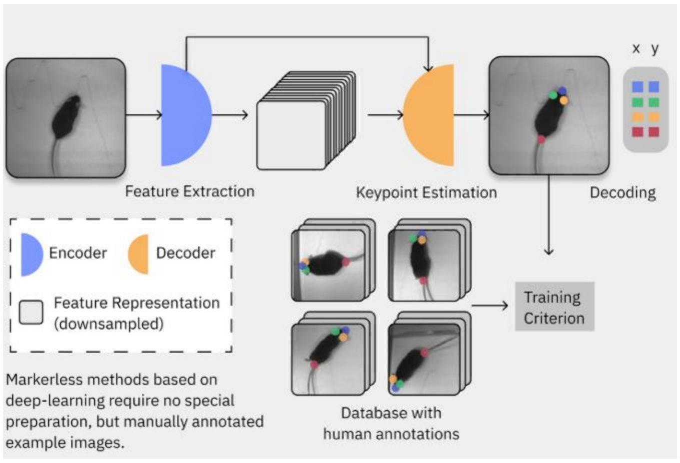

x y
Feature Extraction
Keypoint Estimation
Decoding
Encoder
Decoder
Feature Representation (downsampled)
Training Criterion
Markerless methods based on deep-learning require no special preparation, but manually annotated example images.
Database with human annotations

Figure 1.5: Markerless pose estimation based on deep learning: Using human annotated labels, the ML models are trained to infer keypoint representations directly from the image or video used as input. The architecture of these models usually consists of a feature extractor and a decoder, which are trained end-to-end in modern systems. Adapted from Figure 2. [13]

Figure 1.5 depicts an example of an object detection algorithm, where the images of a mouse are used as input for the feature extraction (in this example also referred to as encoder), followed by the representation of the downsampled features. At the end, there is a keypoint estimation (decoder) which outputs the x and y coordinates for the body parts of interest. Only a database consisting of human-labeled images is used for training, so no special equipment is required. [13]

The data set, the model, the loss function, and the optimization are the four fundamental elements of ML algorithms [16]. Decisions regarding these components affect the quality of performance as well as the behavior of a pose estimation and provide the opportunity to design the system according to the requirements of the application. [13]

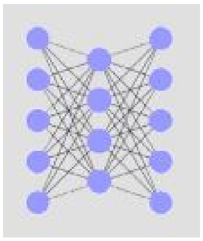


Figure 1.6: Neurons and their layer connections in an ANN: The neurons are arranged in a directed graph, each of them is connected to all neurons in the next layer. Each connection represents a learnable parameter (weights). Adapted from Box 1. [13]

Each model is made up of a particular number of individual layers, which in turn consist of simple units [13]. These units are connected to the units of the next layer in a unique way, depending on the chosen model [11]. In the case of ANN, each neuron is connected to each neuron in the following layer, and each of these connections reflects the model's weights (see Figure 1.6) [13], whereas in CNNs, the layers are connected by filters, which represent the parameters (more details in Section 1.2.2) [17]. The algorithms learn by iteratively updating their weights using the optimization algorithm to minimize the loss function that compares the model's predicted pose to human-labeled truth data, thus serving as a quality indicator [13].

Overview of available packages

To develop proper data loaders, data augmentation pipelines, and training regimes, an understanding of deep learning languages is required. As a result, several packages have emerged in recent years that provide users with a complete pipeline. This starts with custom data labeling, including frame selection and labeling tools, and continues with the creation of training and test data sets using data augmentation and loaders, to neural networks, with code for performance evaluation and tools for reading captured machine-labeled data. [13]

Table 1.2 provides a summary of the most commonly used animal pose estimation tools. As indicated in this table, DLC has many advantages over the other tools, the presence of pre-trained networks being one of the most important ones. Apart from that, DLC is independent of the species, can be used for 3D as well as multi-animal projects, and provides a training code, a GUI, and example data. [13]

Due to all of these advantages, it was decided to use DLC, a CNN based algorithm, for this dog gait analysis application (see Chapter 3: materials and methods).

	Any Species	3D	>1 Animal	Training Code	Full GUI	Ex.Data	PT-NNs	Released	Citations
DeepLabCut	yes	yes	yes	yes	yes	yes	many	4/2018	491
LEAP	yes	no	yes	yes	yes	yes	no	6/2018	98
DeepBehavior	no	yes	yes	no	no	no	no	5/2019	15
DeepPoseKit	yes	no	no	yes	partial	yes	no	8/2019	48
DeepFly3D	no	yes	no	2D only	partial	yes	fly	5/2019	21
FreiPose	no	yes	no	partial	no	yes	no	2/2020	1
Optiflex	yes	no	no	yes	partial	yes	no	5/2020	0

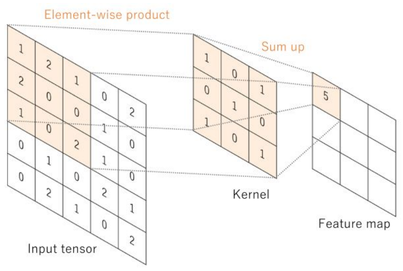
Table 1.2: This table provides a brief overview of popular deep learning tools used for pose estimation in animals. It covers capabilities and benefits, such as whether the tool can be used for each species, for 3D and even multi-animal projects. The availability of a training code, GUI, and example data is also mentioned. Furthermore, the availability of beyond human pre-trained neural networks (PT-NNs) is covered. Adapted from Table 1. [13]

1.2.2. Convolutional Neural Network

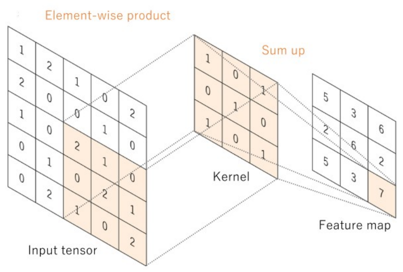
One of the most frequently used tools in the fields of image-related application like object detection are CNNs [18], which show some significant advantages compared to the classical ANNs [19]. One of the biggest benefits of CNN-based models is the much smaller number of parameters, which makes it possible to develop deeper models and thus solve even more complex problems. Moreover, when using CNNs, it is not necessary to care about the position of the object to be detected, the model detects it independent of its position in the images. Higher level features such as faces are thereby identified in the deeper layers, while edges or simpler shapes are detected in the first layers. [19] The layers commonly implemented in CNNs are the convolutional and pooling layers that carry out feature extraction, and the fully connected layer used to map these features to the final output [17].

Convolutional Layer

One of the CNN's fundamental components is the convolutional layer employed to extract significant information from images [18]. The convolution operation is performed between the kernel, also called filter, and the input tensor, which represents an area with the size of the kernel on the input image [17]. For each element in this area, the element-wise multiplication is executed and its results are summed up to create the feature map [18]. This procedure is repeated until the filter has been applied to all areas of the input image, see Figure 1.7. In this step, two hyperparameters can be defined: the kernel size as well as the number of kernels. When applying this procedure to more than one kernel, one obtains a large number of feature maps, each representing different characteristics of the input tensors. [17]



(a) Convolution for the first result value in the feature map



(b) Convolution for the last result value in the feature map

Figure 1.7: Convolutional layer: In this example, the kernel size is 3×3 and the stride is one. The convolution is calculated for each element in the input tensor. The results are summed up and displayed in the feature map. Then, the tensor is moved with a stride size of one and the procedure is repeated for the next area. This process is performed repeatedly until the last area of the input image is reached. Adapted from Figure 3. [17]

Another hyperparameter to be specified is the stride, which describes the distance between two successive kernel positions [17]. Usually, a stride of one is used [17] since applying a small value results in a more detailed representation of the image [18]. In case the stride should provide a downsampling of the feature map, which reduces the computation time, a larger value is required [17]. The use of convolution operations as described above results in a reduction of the height and width of the output feature map, and it is not possible to position the center of the kernel on the outermost element of the input tensor. To counteract these problems, there is a special technique called padding, mostly zero padding, which adds rows and columns around the input tensor filled with zeros, as shown in Figure 1.8. With the use of padding, a convolutional layer has a total of four hyperparameters that must be set before training the model. These include the kernel size, the number of kernels, the stride and the padding. [17]

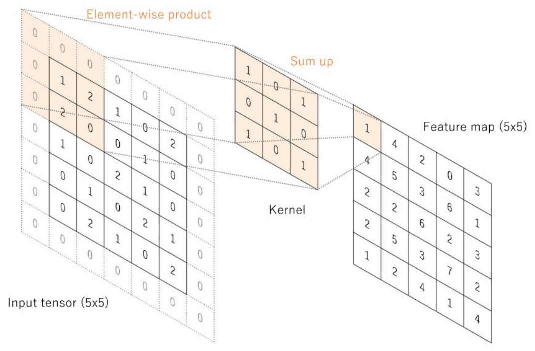


Figure 1.8: Convolutional layer with padding: In this example the kernel size is 3×3, and the padding and stride are set to one. The padding adds one row and one column on each site of the input tensor and increases the size from 5×5 to 7×7. Therefore, the center of the kernel is now also on the outermost element of the input tensor and the feature map size is the same as the input image size, so there is no shrinkage. [17]

Pooling Layer

A very common and frequently used method for downsampling is to use pooling layers as an alternative to employing large strides in the convolutional layer [17]. As already mentioned in the previous section, a high value used for the stride results in a reduced representation of the image, i.e., it is possible that important details of the picture get lost. Therefore, the preferable option is to use a small stride size to gather as much information as possible, then use the pooling layer to reduce the feature map's size by maintaining just the relevant features. [18] Downsampling feature maps reduces their in-plane dimensionality,[3] introduces translation invariance to tiny shifts and distortions, and limits the number of parameters that can be learned. Max pooling is one favourite pooling operation used in CNNs. It extracts a specified patch (defined by the pooling layer's kernel size) from the input image and preserves only the maximum value of this area. [17]

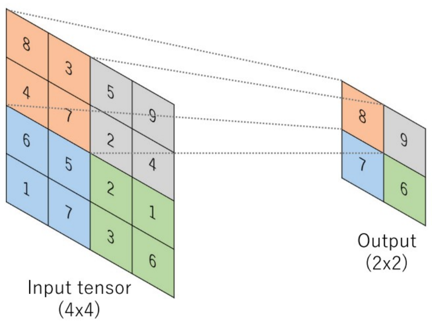


Figure 1.9: Max pooling: It extracts a specified patch, which is defined by the kernel size of the pooling layer (in this case it is set to 2×2) and keeps only the max value of the patch in the feature map to reduce the output dimension. Adapted from Figure 6. [17]

Figure 1.9 shows an example of a max pooling layer with a size of 2×2 and a stride of two. This is a commonly used filter size, as it downsamples the in-plane dimension by a factor of two. A patch with the size of the kernel is extracted on the input tensor and only the maximum value is kept.

[3] Note: Only the height and width are reduced, the depth of the feature maps remains the same.

With a stride of two, the next patch area is extracted and again only the maximum value is kept. This process is repeated until the entire range of the input tensor has been used. In this example, the dimension of the input tensor is 4×4, so the max pool operation is performed exactly four times, resulting in an output dimension of 2×2. [17]

Another type of pooling is average pooling, which calculates the average for the specified area of the input-tensor instead of simply keeping the maximum value [18]. In addition to the effect that, as with max pooling, the learnable parameters are reduced, average pooling ensures that the CNN also takes inputs that vary in size. Pooling layers have four hyperparameters, these include the pooling method, the kernel size, the stride, and the padding, but they do not have any parameters that can be learned. [17]

Fully-Connected Layer

Finally, the features extracted by the convolutional layers and downsampled by the pooling layers are mapped from a subset of fully-connected layers to the final output [17]. As the name implies, in a fully-connected layer, every node is connected to every node in both the previous and the next layers, much like the neurons in ANNs. Each connection represents a learnable weight, which means that in a CNN, fully-connected layers own most of the parameters and thus also incur high computational cost. [19]

Activation functions

The application of activation functions is necessary for a Neural Network to handle complex tasks; otherwise, the network would behave as a Linear Regression Model [20]. Therefore, the output of the convolution operation as well as the output of the fully-connected layers are passed through nonlinear activation functions. There are several types of nonlinear activation functions, the most commonly used are the sigmoid, hyperbolic tangent, rectified linear unit (ReLU), and softmax. [17] The convolutional layer's activation function is selected based on the training speed and the model's performance, while the selection of the activation function applied to the fully-connected layer depends on the model's application [21].

Sigmoid (see Equation 1.1) is applied element by element to the feature map, transforming each element individually to the range between zero and one. During the backpropagation (described in detail in Section 1.2.4) used to update the weights and kernels in the algorithm, derived activation functions must also be propagated. As can be seen from Equation 1.2, which shows the derivative of the sigmoid function, a large input value causes the gradient of sigmoid to be very close to zero, i.e., the weights and kernels are poorly updated, resulting in nearly no learning progress of the algorithm. This problem is known as "vanishing gradient" and becomes more and more significant as the depth of the network increases, because the multiplication of small values along the backpropagation path leads to ever decreasing gradients. This can reach a point where the gradient becomes zero, which means that the weights and kernels are not updated at all. [21]

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.1}$$

$$\frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{1.2}$$

As can be seen in Equations 1.3 and 1.4, the **hyperbolic tangent** function is quite similar to sigmoid. However, the range now extends from minus one to one and the gradient of the hyperbolic tangent is much steeper. Due to the fact, that the function is centered at zero and the gradients are not limited to a specific direction, the hyperbolic tangent is favored over the sigmoid function. [20]

$$tanh(x) = 2sigmoid(2x) - 1 \tag{1.3}$$

$$\frac{d}{dx}tanh(x) = \frac{4e^{-2x}}{(1+e^{-2x})^2} \tag{1.4}$$

With the **ReLU** function represented in the Equations 1.5 and 1.6, the problem of the vanishing gradient is mitigated, since it's derivative is always one for any input value greater than zero. Unfortunately, for negative values or zero values as input, the gradient of the ReLU function becomes zero, so in this case the weights are not updated. This problem is known as "dying ReLU", but there are already various functions derived from the ReLU that counteract this problem. These include the exponential linear unit (ELU) (see Equations 1.7 and 1.8), the leaky ReLU (see Equations 1.9 and 1.10), and the scaled exponential linear unit (SELU) (see Equations 1.11 and 1.12) whose gradients for negative values (or zero values) as input are no longer zero. [21]

$$ReLU(x) = max(0, x) \tag{1.5}$$

$$\frac{d}{dx}ReLU(x) = \{1\ if\ x\ >\ 0;\ 0\ otherwise\} \tag{1.6}$$

$$ELU(x) = \{x\ if\ x\ >\ 0;\ \alpha * (e^x - 1)\ if\ x\ <\ 0\} \tag{1.7}$$

$$\frac{d}{dx}ELU(x) = \{1\ if\ x\ >\ 0;\ (ELU(x) + \alpha)\ otherwise\} \tag{1.8}$$

$$LeakyReLU(x) = \{x\ if\ x\ >\ 0;\ \alpha x\ if\ x\ <\ 0\} \tag{1.9}$$

$$\frac{d}{dx}LeakyReLU(x) = \{1\ if\ x\ >\ 0;\ \alpha\ otherwise\} \tag{1.10}$$

$$SELU(x) = \lambda * \{x\ if\ x\ >\ 0;\ \alpha * (e^x - 1)\ if\ x\ <\ 0\} \tag{1.11}$$

$$\frac{d}{dx}SELU(x) = \lambda * \{1\ if\ x\ >\ 0;\ \alpha * e^x\ otherwise\} \tag{1.12}$$

Equation 1.13 shows the **softmax** function, which is basically used as the activation function for the fully-connected layers in a network. The function normalizes the output values of this last layer to the probabilities of the target classes and is therefore applied for classification tasks with multiple classes. [17]

$$softmax(x)_i = \frac{e^{x_i}}{\sum\limits_{j=1}^{K} e^{x_j}} \tag{1.13}$$

1.2.3. Residual Networks

The principle underlying residual learning connections or "shortcut connections" is simply skipping one or more layers and performing identity mapping[4], which is then added to the output of the stacked

[4] An identity map (also known as identity function) is a function, that always returns the value that was used as its argument. So the input is equal to the output.

layer, as can be seen in Figure 1.10. These shortcut connections do not add any extra parameter or complexity in computation and still allow for an end-to-end training of the entire network using stochastic gradient descent (SGD) with backpropagation. In general, as the number of layers in a network increases, so does the time required for training. To counteract this phenomenon, the building block used in deeper Residual Network (ResNet)s such as ResNet-50/101/152 is modified to a so-called "bottleneck" building block. As visible in the right picture of Figure 1.10, this "bottleneck" design consists of three layers, first a 1×1, followed by a 3×3, and finally a 1×1 convolutional layer again, whereby the 1×1 convolutional layers are responsible for reducing and increasing the depth of the feature maps. [22]

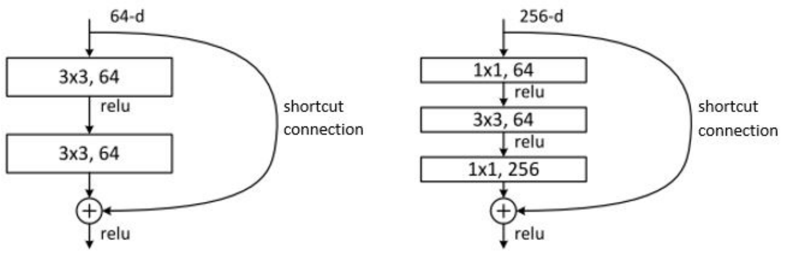


Figure 1.10: Building blocks in ResNets: The left image depicts a building block section from the ResNet-34 architecture. In this case, shortcut connections are added in every second layer. The right image shows a modified building block, the so-called "bottleneck" building block, which is used in deeper ResNets (e.g. 50/101/152 layers). In each shortcut section, 1×1 convolutional layers are used to increase or decrease the depth of the feature maps. [22]

By using this bottleneck design, a 152-layer ResNet, for instance, requires 11.3 billion floating-point operations, whereas a VGG-16 network (16 layers, no shortcut connections), requires 15.3 billion. This contrast demonstrates that, despite the significant layer increase, ResNets are less complex than networks without shortcut connections. [22] In other words, by employing residual learning connections, a network's depth can be increased without sacrificing performance. Another significant benefit of ResNets is, that identity mapping holds true not only during forward propagation but also during backpropagation and thus prevents the occurrence of vanishing or exploding gradients (explained in detail in Section 1.2.4). [23]

1.2.4. Network training

Training a network is an iterative process with the goal to find the most suitable model for the application at hand. To achieve this, the algorithm updates its weights or kernels until the discrepancy between the output predictions and the ground truth labels of the training data set reaches its minimum [17]. The optimization algorithm most frequently used for this purpose is the gradient descent (GD) [18]. The basic steps for this process are as follows: The algorithm takes the input and predicts an output using its current weights and kernels. The loss function compares these predictions to the labeled ground truth data and calculates its loss value. Finally, the weights or kernels get updated via GD and backpropagation according to the calculated value and the process starts over again. This process is shown schematically in Figure 1.11, using a CNN in this example. [17]

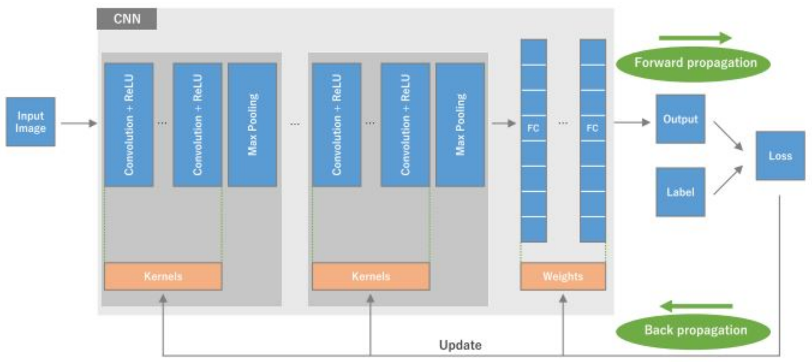


Figure 1.11: Training process of neural networks: The algorithm takes the input image and predicts the output by using its current weights and kernels. This output is compared to the labeled data using the loss function, and the loss value is calculated. According to this value, the kernels and weights are updated and the process starts over again. [17]

Loss function, learning rate and gradient descent

The type of loss function belongs to the hyperparameters and is thus chosen depending on the particular application. The cross-entropy function, for example, is frequently employed in multiclass classification, while the mean square function is more generally utilized in regression applications. To minimize the estimated loss value, the negative direction of the gradient of the loss function is used to update the weights and kernels. [17] This gradient is calculated with respect to all weights within the network, and after all gradients have been computed, each weight gets updated accordingly. This entire process of updating the network weights using GD in order to reduce the loss is called backpropagation. [18] Equation 1.14 provides the mathematical expression for updating a single parameter, where w stands for the weight, α for the learning rate and L for the loss function [17].

$$w := w - \alpha * \frac{\delta L}{\delta w} \tag{1.14}$$

To indicate the step size of the update, the gradient is multiplied by the learning rate α, which is another hyperparameter of the network [17]. The significance of choosing this value[5] is demonstrated in Figure 1.12. The thick black dots represent the starting weight, i.e., the weight with which the algorithm is initialized. As already mentioned, the target is to reduce the loss to a minimum, in terms of the graph this means to reach the global minimum. As can be seen in the right image of Figure 1.12, choosing a small value for alpha leads to a short step size, making the GD very slow. By choosing a high learning rate, one can accelerate GD, but there is a chance that doing so will cause the algorithm to become stuck in a local minimum and fail to reach the global minimum (see left picture). [18]

For the GD method, the network weights are always updated after all training data has passed through the network once. In other words, if for example one million data samples are used for the training process, the initial update of the parameters is performed after one million samples have been sent through the model. Before the weights are updated the second time, the entire data must again traverse the network, and so forth. Since it takes more than one update of the parameters to optimize a model, i.e., to reach the minimum loss, the GD method can lead to extensive training times, especially for large training data sets. To counteract this, the SGD method, in which the weights are updated after each single data point, is frequently applied. In contrast to the GD method, the

[5] Note: The values to be considered for the learning rate range from 10^{-6} to 1.

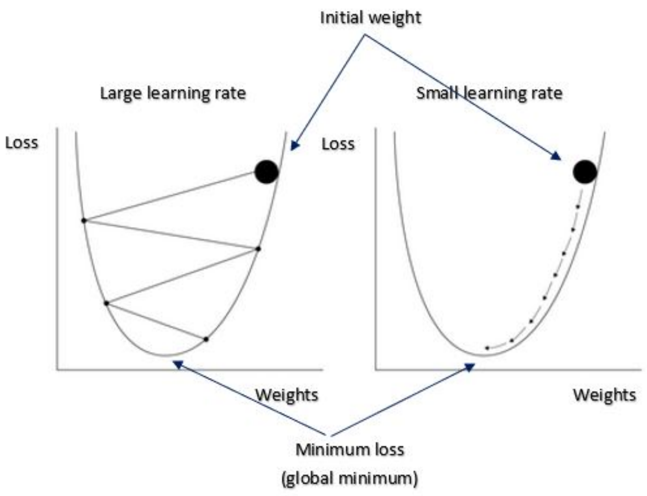


Figure 1.12: High learning rate vs low learning rate: This figure illustrates the loss function as a graph of loss to weights, where the black dots represent the initialized weight. Updating the weights with a large α leads to big steps downward, i.e., the GD is fast but could also prevent the reaching of the global minimum (left image). To make short steps a small learning rate is necessary, but this also slows down the GD as shown in the right picture. Adapted from Figure 4. [18]

model already learns with the first single data point, which makes it converge faster and reduces the required training time. The technique known as mini-batch GD represents a combination of SGD and GD. With this method, the user-defined hyperparameter called batch-size is used to divide the training data set into subgroups. After one subset has fully traversed the network, the weights are updated. If the batch-size is set to 50, for instance, the first update occurs after 50 data points have been processed by the model. [18]

As already briefly described in previous sections, using gradient descent and backpropagation, especially in deeper networks, may cause the problem of **vanishing and exploding gradients**. The reason for this is that each layer depends on its previous layer, so when computing the derivatives of the loss with respect to the weights, it is necessary to propagate back through the network to the very first layer. The various derivatives are multiplied throughout this process, resulting in ever-increasing values (exploding gradients) for big values and ever-decreasing values for small values (vanishing gradients). As already described in Section 1.2.2, the vanishing gradients may be caused by the use of sigmoid or tanh as activation functions in the layers and can be solved by replacing these functions with ReLU. [18]

The problem of the exploding gradients occurs when the network is initialized with large weights and can be circumvented by means of the **gradient clipping** technique. In this method, the gradients are normalized according to a vector norm and limited to a specified range. When a gradient value exceeds this range, it is replaced by the boundary value of the range closest to this value. For example, if the range is limited to ± 0.7 and the gradient exceeds it in the positive direction, it will be replaced with $+0.7$ and vice versa. [18]

<u>Data set</u>

The data used to train and test the algorithm is essential to the success of a deep learning application because it gives the model experience. At the beginning of a model's training, it does not yet know how to perform well on that particular task. By repeatedly being exposed to data, the model gains an understanding of the data and can continuously improve its performance. [16] Before the training starts, the entire data set is at least split into a training set and a test set [13]. Depending on the task, it might be useful to additionally use a validation or development[6] set [17]. For the optimization of the model's parameter during the training process, only data from the training data set is used [16]. Accordingly, it is necessary that the amount of data in the training set is much larger than in the test or validation set. This fine-tuning process involves evaluating the model after certain training iterations before adjusting the hyperparameters and restarting the training. This is where the validation or development data comes in, as the evaluation needs to be performed on data never used during the training. Depending on the application, the training procedure may require multiple evaluations, so the model could start to overfit to the validation data. [17] Therefore, only the test data set, which comprises data that have not been exposed in either the training or the evaluation process, is used to diligently examine whether the trained model generalizes effectively and so offers outstanding results even on basis of new data [16].

Overfitting and techniques to prevent it

Overfitting occurs when the trained model does not generalize successfully from the training data to new data, i.e., the model works perfectly on the training data but does very badly on unseen data. This issue typically occurs when the training set is too small, has insufficiently accurate data, or contains excessive noise, so it comes to noise learning. [24] Utilizing overfitted models has some drawbacks. Models that are overfitted demand more memory and computing power than models that are not overfitted. Additionally, employing such models might need gathering extra (unnecessary) features for every instance, raising the cost and complexity of predictions. A medical diagnosis made with an overfitting model, for instance, would need pointless testing. Overfitted models may perform less accurately on new data than models that are not overfitted, which is another drawback. This impact has been proven in numerous systems and areas. [25]

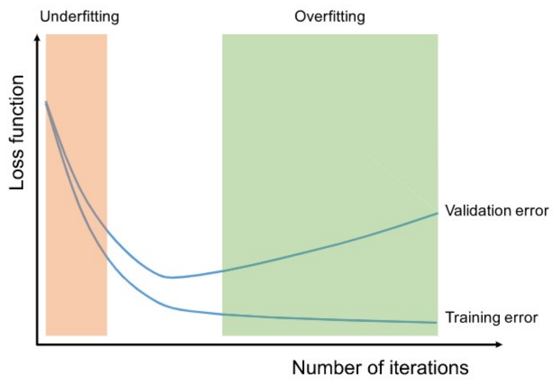


Figure 1.13: Over- and underfitting: Recording the loss over the iterations during the training and validation process could help to detect overfitting or underfitting of the model. If the model performs badly in both cases, i.e. both errors are high, underfitting is present and further training is required (red colored area). Overfitting may be identified if the gap between the validation and training errors is large (green colored area), i.e., the model performs well on the training data but poorly on the validation data (unseen data). [17]

Possible overfitting or underfitting of the model can be detected by keeping track of the losses during the training and validation phases. As can be seen in Figure 1.13, underfitting (red colored area) occurs when the algorithm performs poorly in both cases. This underfitting indicates that the network has been under-trained and can be solved by additional training. In case of overfitting, the training error is already very small while the validation error is high, in other words, there is a relatively large gap between the two. [17] So the objective is to find the moment when there is a perfect

[6] The "validation" in medicine corresponds to the "test" in ML; to avoid confusion, the word "development set" is sometimes used instead of "validation set".

balance between overfitting and underfitting. In addition to monitoring the loss, keeping track of the accuracy on the test data can help to achieve this goal. If the accuracy does not improve with further training, the training should be stopped. [24]

Since a model trained on large data sets usually generalizes better, one solution to reduce overfitting of the algorithm would be to increase the amount of training data. Due to the fact that additional data is often not available, other techniques such as regularization with dropout or weight loss, batch normalization, and data augmentation have been developed. **Regularization with dropout** ensures that the model is less sensitive to certain weights by setting randomly picked activations to zero during training, while the **weight decay** technique penalizes the model's weights so that they take only small values. **Batch normalization** is a sort of flexible layer that adaptively normalizes the input values of the subsequent layer. Aside from the effect of reducing overfitting, batch normalization also makes it possible to use greater learning rates and a lower dependence on initialization. With the help of **data augmentation**, the training data set can be expanded by adding transformed images. For this purpose, the images from the original training set are randomly modified by cropping, rotating or mirroring them, for example. [17]

Transfer Learning

In many applications, researchers face the problem that the data set used is quite small, but training a model from scratch requires a lot of data. By means of transfer learning, however, it is possible to train a model even with small data sets and still achieve outstanding performance [26]. The fundamental concept behind transfer learning is to use a model that has already been trained as a jumping off point for a comparable problem in a different but related field [27]. In other words, the network for the new task is initialized with the weights from the network that was trained from scratch with a huge data set [13]. In this process, the fully connected layers of the pre-trained model must be replaced by the fully connected layers applied for the new task, leaving only the convolutional base (convolutional layer and pooling layer) of the transferred model. Now, the model for the new task can either be trained using the fine-tuning method, where all or parts[7] of the weights, including the ones of the convolutional base, are updated, or using just the weights of the newly added fully-connected layers (fixed feature extraction method). [17] The use of pre-trained models is highly advisable when only a small amount of labeled data is available for the actual task, since less training data is needed, and it also saves training time and increases robustness. [13]

1.3. Classification

The purpose of classifiers in medicine, the basic structure of decision tree-based models, and the difference between Random Forest (RF) and XGBoost are briefly explained in this section.

1.3.1. Decision tree based classifiers

A decision tree based classifier makes predictions by splitting the input as long as every data point is classified [28]. The decision tree's basic structure is shown in Image (a) of Figure 1.14. Except for the initial point at the top of the tree, which is referred to as the root, the split points of a decision tree are called nodes. [28]. All splits result in a specific number of branches, each of which contains the corresponding subset of the data. The following nodes use these subsets for decision-making and further divide them in accordance. This splitting process is performed until the end of the tree is reached and each record is assigned to a class. The nodes at the end are called leaves and contain all final predictions. [29]

[7] Note: Typically, the layers at the beginning of the network are not updated because they extract more general features like edges and lines.

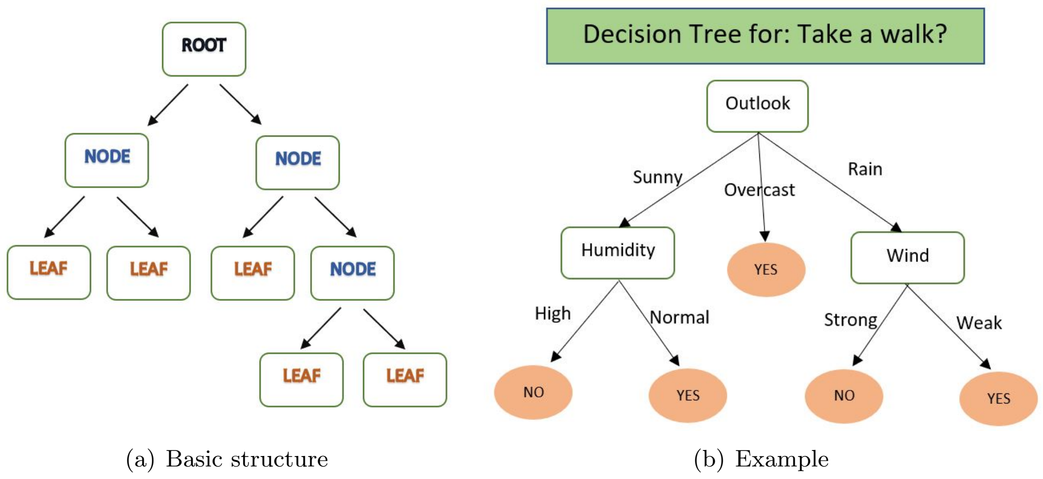
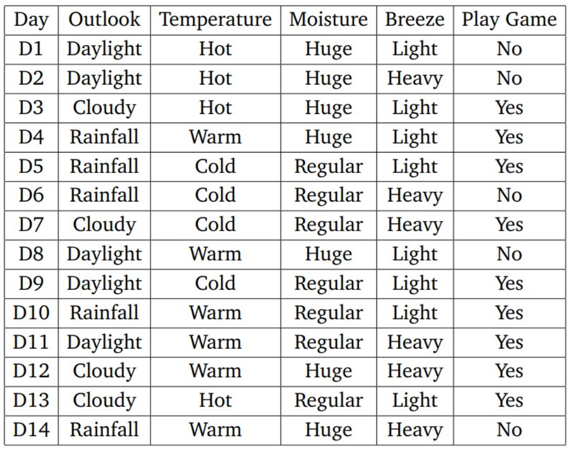


Figure 1.14: Basic structure and an example of a decision tree: The left picture (a) represents a basic structure. At the top of the tree is the root, where the first split is carried out. The arrows represent the branches that lead to the nodes. The nodes at the end of a tree are called leaves and contain all final predictions. Image (b) shows a decision tree example for a task called "Take a walk?", where "Outlook", "Humidity", and "Wind" represent the feature vectors on which the tree makes decisions. The corresponding attribute values of the vectors are visible on the branches, e.g., "High" and "Normal" are attributes of the "Humidity" vector. The circles represent the leaves, containing the class labels; in this example the classes "YES" and "NO".

An example of a decision tree model is given in image (b) of Figure 1.14. Based on the attribute-value vectors "Outlook", "Humidity", and "Wind", the model decides whether to take a walk or not, i.e., whether the class is "YES" or "NO". The attribute values for each of these vectors are represented by the corresponding branches, e.g., "High" and "Normal" are the attribute values of the vector "Humidity". The decision tree attempts to learn how to categorize objects by examining a collection of instances [30]. An instance is a case, whose class is already known [30], i.e., for the presented example, one of the cases might be as follows: Outlook = Sunny, Humidity = Normal, Wind = Weak. If the class, in this example it would be the class "YES", is already known for this case, it is called an instance. Decision trees use these instances, which are typically represented as attribute-value vectors as demonstrated in the example, as learning input and produce mappings from attribute values to classes. After adequate training, the mapping should be able to reliably classify both the given cases and new (unknown) cases. [30]

The aim of every decision made within a tree is to find a data split that results in the least possible error. This is accomplished by using an error method known as the "gini criterion", which determines the gini index for each potential split. This index lies between zero (zero errors) and one (all errors), i.e., the closer this index is to zero, the smaller the error. An index of 0.5, for example, indicates that the elements are distributed evenly, which is no better than a random estimate. Equation 1.15 shows the mathematical expression of the gini criterion, where p_i stands for the probability that the split leads to the right value and the total number of classes is provided by *c*. [29] Before the algorithm picks a decision node, first the gini index is computed for each attribute followed by calculating the weighted sum of the gini indexes for the feature. Based on these outcomes, the attribute with the lowest gini index value is picked. [31]

$$\mathit{gini} = 1 - \sum_{i=1}^{c} (p_i)^2 \tag{1.15}$$



Day	Outlook	Temperature	Moisture	Breeze	Play Game
D1	Daylight	Hot	Huge	Light	No
D2	Daylight	Hot	Huge	Heavy	No
D3	Cloudy	Hot	Huge	Light	Yes
D4	Rainfall	Warm	Huge	Light	Yes
D5	Rainfall	Cold	Regular	Light	Yes
D6	Rainfall	Cold	Regular	Heavy	No
D7	Cloudy	Cold	Regular	Heavy	Yes
D8	Daylight	Warm	Huge	Light	No
D9	Daylight	Cold	Regular	Light	Yes
D10	Rainfall	Warm	Regular	Light	Yes
D11	Daylight	Warm	Regular	Heavy	Yes
D12	Cloudy	Warm	Huge	Heavy	Yes
D13	Cloudy	Hot	Regular	Light	Yes
D14	Rainfall	Warm	Huge	Heavy	No

Figure 1.15: Example data set for gini calculation: The data set consists of the four features "Outlook", "Temperature", "Moisture", and "Breeze" and covers 14 data entries. The column "Play Game" represents the target column with the two classes "Yes" and "No". [31]

For example, for the data set depicted in Figure 1.15, which consists of four features and 14 data entries (rows), first the gini index for the attributes of the feature "Outlook" are calculated (see Equations 1.16, 1.17, and 1.18) and the results are used in the second step to compute the weighted sum for this feature (see Equation 1.19). These steps are repeated for the next three features of the data set, and finally the feature with the smallest gini index is selected to be the first decision node, the root. Thus, the initial partitioning of the tree is based on the Outlook feature, resulting in the three branches Daylight, Cloudy, and Rainfall with corresponding sub-data sets. In the next step, the gini index calculations for the remaining features (the already used Outlook feature is excluded) are performed in the individual branches, using the "newly formed" sub-data sets. These steps are repeated until the tree has been created. [31]

$$\textit{gini(Outlook=Daylight)} = 1 - \sum_{i=1}^{c}(p_i)^2 = 1 - (2/5)^2 - (3/5)^2 = 0.48 \tag{1.16}$$

$$\textit{gini(Outlook=Cloudy)} = 1 - \sum_{i=1}^{c}(p_i)^2 = 1 - (4/4)^2 - (0/4)^2 = 0 \tag{1.17}$$

$$\textit{gini(Outlook=Rainfall)} = 1 - \sum_{i=1}^{c}(p_i)^2 = 1 - (3/5)^2 - (2/5)^2 = 0.48 \tag{1.18}$$

$$\textit{gini(Outlook)} = (5/14) * 0.48 + (4/14) * 0 + (5/14) * 0.48 = 0.342 \tag{1.19}$$

The data used as input for a decision tree must be prepared accordingly, depending on the model. Loading, cleaning, analyzing, and manipulating are among the phases of data preprocessing, all of which are subsumed under the term "data wrangling" [29]. Decision trees tend to overfit, especially when the available training data may not be representative of the population they are supposed to represent. This is because the trees try to refine the training data into subsets that contain just one class. [30] One way to avoid this is to fine-tune hyperparameters accordingly, the other way is to aggregate the predictions of many trees, as is the case with RF and XGBoost classifiers. [29]

Regardless of the area of application, proper decision making is one of the most important key factors in medicine and is generally based on past experience with comparable situations, new study results, and personal judgment. As the number of solved cases and study outcomes is growing rapidly, it is becoming increasingly challenging to manage these huge amounts of data and make decisions. Therefore, more and more technical tools are being used to assist physicians in making decisions. Decision tree based classifiers have already been used widely and successfully in medicine because they have the ability to learn automatically, are simple, and make decisions effectively and reliably. [30]

1.3.2. Random Forest vs. XGBoost

XGBoost (the XG stands for extreme gradient) and RF classifiers belong to the ensemble methods, which are made up of different ML models called base learners that operate together, making them more powerful and less error-prone. The final result is determined by majority vote, meaning that if two out of three models predict NO, the final prediction is NO. The most popular base learners for both are decision trees, the only difference being that RF combines the trees by bagging and XGBoost combines them by boosting. [29]

Bagging is an acronym for bootstrap aggregation, which refers to sampling with replacement. The strategy of bagging is to generate "new" data sets for each of the decision trees in the ensemble by randomly selecting data from the original data set. [28] The process for this is as follows: one single data sample is randomly selected, added to the new data set and returned instantly to the original data. Only then the next sample is randomly selected, giving the option of picking the same sample again. This process is repeated until each new data set contains the same number of data as the original data. [29]

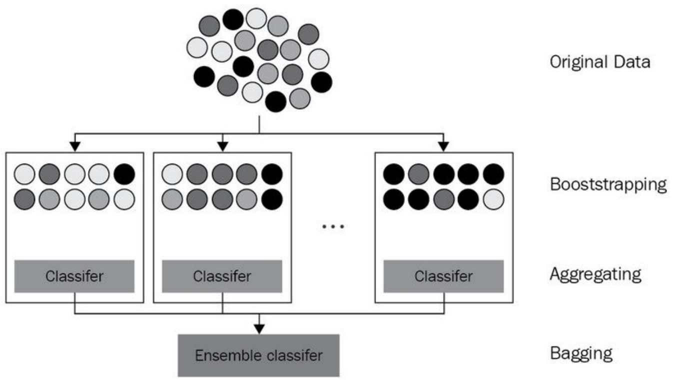


Figure 1.16: Example of bagging in RF classifiers: The original data is represented by 20 marbles. Bootstrapping describes the step in which individual marbles are selected for each decision tree and placed back immediately, resulting in the possibility that some of the marbles are picked more than once, while others are not selected at all. This is evident in the last decision tree, which contains seven black marbles, although only five are included in the data. At the end, the individual results are aggregated and the decision is made via majority rule. [29]

An illustration of the bagging method is shown in Figure 1.16, with the original data set represented as 20 separate marbles. For each individual decision tree, marbles are picked one at a time and returned immediately, thus some may be selected many times and others may not be selected at all. For example, the data set for the last decision tree in the picture contains seven black marbles even though there are only five in the original data set. In contrast, decision tree number one on the left includes only one of the five available black marbles. As already mentioned, the data sets for each decision tree created in RF contain the same number of samples as the original data, so mathematically estimated two-thirds of the samples for each tree are unique and one-third contain duplicates.

Following the model's bootstrapping phase, each decision tree generates its own prediction, which is then aggregated to the final result using the majority rules, as previously indicated. [29]

This bagging method makes the RF an effective classifier with many advantages. Apart from the fact that RF models are faster than AdaBoost models (these models will be explained in the following sections) without losing accuracy, they are not susceptible to overfitting. Moreover, RF is more noise resistant due to the combination of bagging and random selection of features to split, and it can handle both continuous and categorical variables. Another major advantage is the so-called out-of-bag (OOB) data which is generated during bagging. [32] By means of a hyperparameter, it can be specified that the data remaining after the bagging process is retained as OOB data [29]. With this data, the generalization error, correlation and strength, and the importance of individual variables can be estimated without the need of a separate test data set [32].

Over the past few decades, **boosting** has been one of the most effective general machine learning algorithms for attaining optimal outcomes. The idea behind boosting is to learn from the previous tree's errors and create a new tree on top of it that corrects those errors. This is the main difference to the bagging method, where each tree is built from scratch. In this learning process, algorithms transform weak learners, which are slightly better than random guessing, into strong learners. To benefit from this repeated error correction, it is important to start with weak learners, as using strong learners from the beginning will restrict the learning process. [29]

Boosting may be divided into two types: the **AdaBoost** method and the **gradient boosting** method. AdaBoost is one of the early boosting models and focuses only on updating the weights that have a greater impact on erroneous predictions. [29] To do this, AdaBoost models fit the weak learners to the data set, weight the sum of each weak learner's predictions, take the one with the fewest classification mistakes and update this learner's weights focusing on those most likely to have contributed to an incorrect prediction [28]. The gradient boosting method builds the new tree also based on the errors of the previous tree, but here already correct decisions are disregarded. The basic steps for the training process of an gradient boosting model are as follows: First step is to initialize a weak base learner and fit it to the data set. After that, predictions are made using the training data and the residuals[8] are calculated. These residuals are used as target column for the next tree, i.e., the new tree is fit to the calculated residuals and not to the training data and each weight is updated accordingly. With repeating this process, the residuals get smaller and smaller and the learners become stronger. When the training is finished, predictions are made for each tree in the ensemble using the test data set and the prediction results are summed up and used for the evaluation of the model. [29]

An advanced variant of gradient boosting is the open source **XGBoost** model [33], where the "extreme" in extreme gradient boosting refers to pushing the boundaries of computational performance to their farthest limit [29]. Thanks to the scalability of the XGBoost models, state-of-the-art results can be achieved for a wide range of problems using far fewer resources. This scalability is due to various algorithm and system improvements, such as sparsity-aware split finding, parallel processing, cache access, and block compression and sharding. [33] The sparse-aware split finding technique allows only sparse matrices for identifying the splits within the tree. This makes XGBoost much faster than competing systems, since only non-zero data points are stored in these matrices, saving a significant amount of space. [29] Since data sorting is the most time-consuming step in learning with trees, XGBoost employs in-memory units called blocks. Each column is sorted by the feature value, and the data for each block is saved in compressed column format. By utilizing these blocks, XGBoost is able to employ parallel computing for both the sorting and split finding processes, which ultimately saves a significant amount of time. [33] With all these improvements, XGBoost models run more than 10 times faster on a single computer than other systems [33], so it was decided to use the XGBoost classifier for the gait analysis application (see Chapter 3: materials and methods).

[8] Difference between the model's predictions and the actual values.

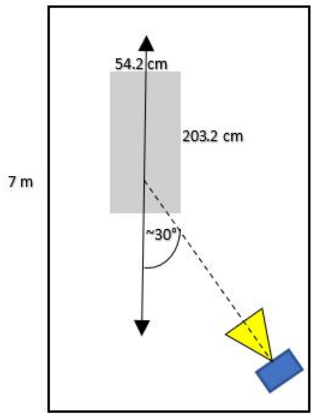
2. Aims and objectives

Develop a Python-based project that uses state-of-the-art ML techniques to observe a dog's gait based on a video and measured pressure data and detect any irregularities. If the dog is lame, the program is supposed to classify the dog as diseased, otherwise as healthy.

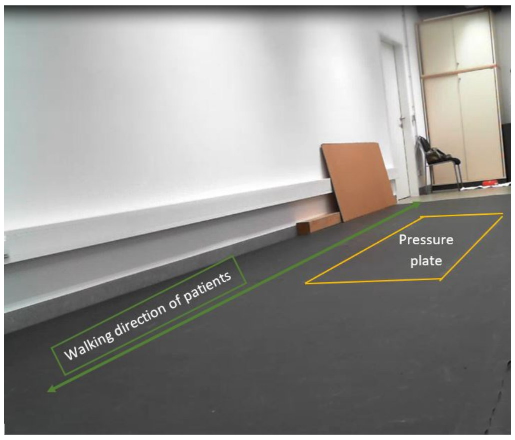
3. Materials and methods

3.1. Provided data

The data used for the training, evaluation, and testing process was provided by VetMed and covers twelve different dogs, six of whom are diagnosed as healthy and six as lame. For these gait analyses, the patients were guided over a pressure plate in a laboratory environment. Picture (b) in Figure 3.1 shows a screenshot taken from one of the provided videos illustrating the laboratory situation from the camera perspective. To prevent the dogs from being distracted or slipping, the pressure plate (marked with a yellow rectangle) was covered with a rubber mat and embedded in a 7 m long walkway [1]. The green arrow in Figure 3.1 indicates the walking direction, i.e., the patients walked towards and away from the camera obliquely. The image on the left shows the laboratory environment in abstract form: the gray rectangle represents the pressure plate (54.2 x 203.2 cm) and the yellow triangle together with the blue rectangle represent the camera, which was placed at a 30° angle to the pressure plate.



(a) Abstract picture of laboratory environment



(b) Real laboratory environment

Figure 3.1: Laboratory environment for gait analyses: The pressure plate, marked with a yellow rectangle, is mounted in a 7 m walkway and covered with a rubber mat to avoid slipping and distraction of the patients [1]. The green arrow indicates the walking direction of the patients, which was more or less diagonally towards and away from the camera. The picture on the left shows an abstract interpretation of the laboratory environment. The grey rectangle represents the mounted pressure plate. The camera was positioned in a 30° angle to the walking direction.

As already briefly described in Section 1.1.2 "The Pressure Analyzer", the measured data was processed with the PA tool and finally exported to an Excel file. Thus, data is stored in two different formats for each patient, namely the ppf files, which can be read and displayed with the PA, and the associated xlsx files. In addition to the collected data, each gait analysis was recorded and saved on video. To make a diagnosis, the dog was observed during both walking and trotting [3], which is why a total of two independent gait analyses were performed per patient. Separate folders were provided for each patient, whereby each of these folders contains four data and two video files of the respective patient. As a result, a total of 24 Excel files with the measurement data and 24 gait analysis videos were available for training, evaluation, and testing.

Naming of the files

In order to distinguish between the different patients as well as the two gaits, step/walk and trot, all files were named according to a specific scheme. As can be seen in Table 3.1, a simple numbering from 01 to 12 is used to differentiate between the patients. This is followed by adding an "H" for a dog diagnosed as healthy or a "KJ" for a lame dog. Next, the differentiation of the gaits was carried out by specifying an "S" for the step/walk or a "T" for the trot. With the numbering at the end of the file name, the gait analysis processes and thus the recorded videos are numbered consecutively[1]. Files belonging to one patient and one gait analysis, i.e., step/walk or trot, differ only in their file types (ppf, xlsx, mkv or avi). For example, **P0001_H_S_V0001.mkv** represents the video file recorded during the step/walk of patient number one, who is diagnosed as healthy. The Excel file for patient number twelve, who was analyzed regarding the trot and diagnosed as diseased, is saved as **P0012_KJ_T_V0022.xlsx**.

P00ww_x_y_V00zz			
ww	Patient number	01 - 12	
x	Diagnosis	H	Healthy
		KJ	Diseased
y	Gait	S	Step/walk
		T	Trot
zz	Video number	01 - 22	

Table 3.1: Naming scheme of the provided files.

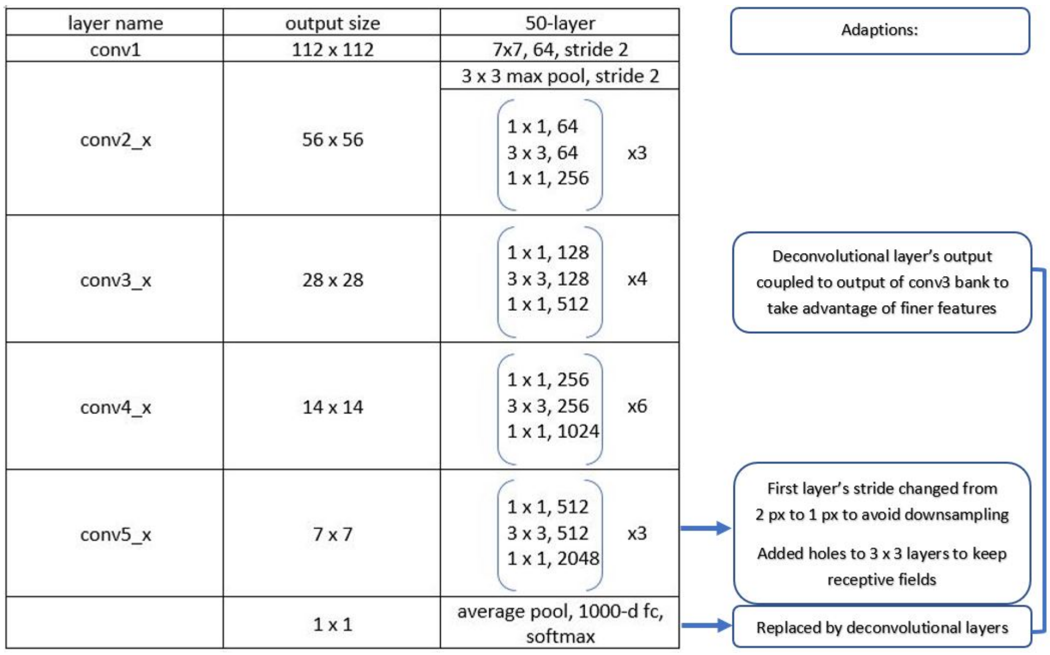
3.2. Body part predictions with DLC

In this section, the DLC algorithm used for the body part predictions is explained, covering its architecture and the training process as well as the parameter settings used for the training iterations.

3.2.1. Architecture

DLC is an open source Python-coded toolkit that allows to train a network based on human labeled data and to automatically label new data [9]. In order to mark the body parts of interest by hand, an interactive graphical user interface is provided through which the extracted frames are loaded, labeled, and saved [34]. In addition, a selection of pre-trained DLC models are available on the "model zoo" website, which can be downloaded and used [26]. The DLC algorithm is based on the part detectors of the DeeperCut algorithm [9], which are built on the 50-, 101-, and 152-layers ResNets [22], which achieve excellent results in detecting human body parts due to pre-training on the ImageNet data set [35]. With over 15 million annotated high-resolution images, which belong to approximately 22.000 categories [36], ImageNet provides a comprehensive data set for object recognition [9]. For the body part prediction, the ResNet-50 architecture described below was used.

[1] Note: The video numbering for patient twelve is the same as for patient eleven, i.e., 24 videos were provided, but the numbering of these videos only ranges from 01 to 22, as 21 and 22 are used twice.



layer name	output size	50-layer
conv1	112 x 112	7x7, 64, stride 2
		3 x 3 max pool, stride 2
conv2_x	56 x 56	[1 x 1, 64; 3 x 3, 64; 1 x 1, 256] x3
conv3_x	28 x 28	[1 x 1, 128; 3 x 3, 128; 1 x 1, 512] x4
conv4_x	14 x 14	[1 x 1, 256; 3 x 3, 256; 1 x 1, 1024] x6
conv5_x	7 x 7	[1 x 1, 512; 3 x 3, 512; 1 x 1, 2048] x3
	1 x 1	average pool, 1000-d fc, softmax

Adaptions:

Deconvolutional layer's output coupled to output of conv3 bank to take advantage of finer features

First layer's stride changed from 2 px to 1 px to avoid downsampling

Added holes to 3 x 3 layers to keep receptive fields

Replaced by deconvolutional layers

Figure 3.2: Genuine ResNet-50 architecture and the adaptions made for DeeperCut: The brackets enclose the respective building blocks, whereby the number next to them indicates the number of stacked blocks. Conv1 consists of a convolution with a kernel size of 7×7 and 64 filters, all with a stride of two, representing one layer, while the Conv2 bank provides three layers per block and three blocks stacked, resulting in a total of $3 \times 3 = 9$ layers. Continuing this way and summing up all the layers, the result is exactly 50 layers. The max/average pooling layers as well as the activation functions are generally not considered, while the softmax and the fc layers are seen as one common layer. DeeperCut has slightly adjusted this architecture for the body part detection. These adaptions are summarized in the blue text boxes on the right. Adapted from Table 1. [22]

As illustrated in Figure 3.2, the genuine 50-layer ResNet consists of five different convolutional banks (conv1 to conv5) and a final layer, where the average pooling, the 1000-dimensional fully connected (fc) layer, and a softmax layer are counted as a common layer [22]. For the sliding-window based body part detection, DeeperCut has slightly adapted the ResNets used (see blue boxes on the right in Figure 3.2). First, both classification and average pooling were removed, then the stride of the respective first convolutional layer in the conv5 block was reduced from 2 px to 1 px to avoid downsampling [35]. To retain the receptive fields[2] of the 3×3 convolutional layers in the conv5 block, holes were added to these layers and deconvolutional layers were finally added at the end of the architecture [35] to generate a scalar field of activation values that correspond to the original image's area [9]. The output of the deconvolutional layer is coupled to the output of the conv3 bank to take advantage of finer features extracted earlier in the architecture [9].

The probability that a body part is at a particular pixel (px) is given by the probability score-maps, whereby each body part has its own sore-map output layer. These score-maps are labeled one (unity probability) during the training process for any body part that is up to 17 px away from the ground truth (human label). To minimize the cross-entropy loss between the predicted and the ground truth score-map, SGD is used. [9] The sigmoid function is employed as the activation functions of the DeeperCut algorithm [35].

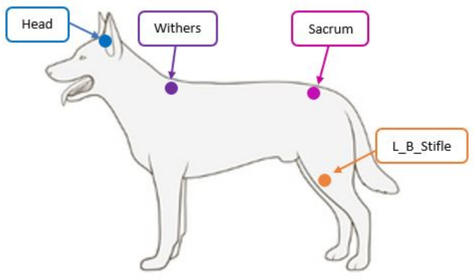
[2] The receptive field represents the size of the region in the input that produces the feature.

3.2.2. Training process

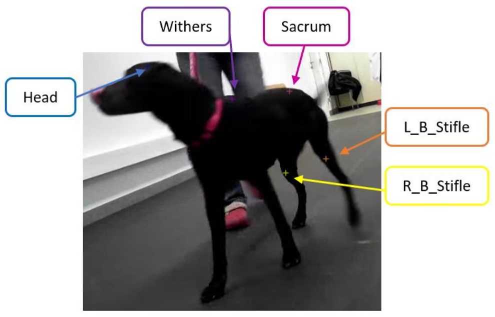
Note: DLC refers to both the passes during a training exercise as well as the number of trainings executed as an "iteration." To avoid confusion, the iterations used to count the number of training sessions performed are termed "training iterations" and those performed during training sessions are termed "internal iterations."

The training process is an iterative procedure in which an evaluation is performed after each training iteration to check the performance of the trained model. If this is not sufficient, the training parameters or the data set are adjusted and the next training iteration is started. This process is repeated until the result of the evaluation is satisfactory with respect to the amount of data available for training. For parameters not explicitly mentioned in the following sections, the default (recommended) parameters of DLC were used.

For canine gait analysis, a total of seven training iterations were performed and evaluated, using a batch size of one to allow different sizes of the frames. Initially, the videos of patients number two through five and eight through eleven were used. For training iterations zero and one, 30 frames were extracted from each of these videos, resulting in a total of 480 frames. For training iterations two and three, ten additional frames per video were extracted, increasing the data set to 640 frames. With iteration four, patient numbers one and twelve were added to further expand the data set. For each of the four added videos, 40 frames were extracted, so that the data set eventually consisted of 800 labeled frames. After training iteration four (part two), a network refinement was performed in which outlier frames were extracted from three videos and added to the data set, providing a total of 1160 frames for training iteration five. The number of frames finally used in each iteration is summarized in the last column of Table 3.2. All extractions were done using "kmean", where DLC first downsamples the video, groups the frames, and then picks frames from different groups to ensure a variety of the training data [34].



(a) The body parts of interest



(b) Example of a labeled frame

Figure 3.3: Labeled body parts of interest: The right picture illustrates a frame extracted from the video file of patient two (P0002_H_S_V0003), showing the five body parts labeled by hand. The abstract image on the left shows a dog from its side, pointing out the positions of the body parts of interest. Since it is the left side, the right back stifle is not visible and therefore not marked.

In the extracted frames, the required body parts, i.e., head, neck, sacrum, and the right and left back stifles, were labeled manually, with only visible body parts marked and hidden ones omitted. The right picture in Figure 3.3 depicts an example frame where all visible body parts are labeled (symbolized by the colored crosses)[3]. The abstract picture on the left serves to illustrate the positions of the placed labels more clearly.

[3] Note: The colored rectangles with the names were added manually for better visibility of the body parts.

For the training iterations, training data sets were created with a train-test split of 0.95, meaning that 95% of the data was used for training and 5% was used to evaluate the models. Before splitting the data, DLC combines all the frames and shuffles them [34]. Training iteration zero was initialized with the weights of the pre-trained model "full dog" (see Table 3.2) downloaded from the open source model zoo [26]. Since a total of 20 body parts were used for this model, but only five are needed for the gait analysis application, the weights for the last layer, i.e., the deconvolutional layer, had to be deleted. Otherwise, DLC is currently not able to process a change in the number of body parts.

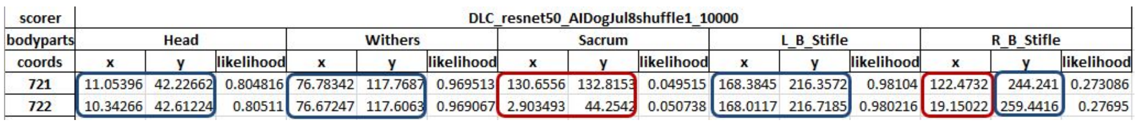
To be able to use weights from previous training iterations, DLC stores them as snapshots with the corresponding index [34]. For each training iteration, snapshots were saved after every 1000 internal iterations. For memory reasons and due to the fact that only the last few snapshots are relevant anyway, a maximum of five snapshots were kept at the same time, meaning that after 5000 internal iterations, the maximum of five snapshots is reached and the algorithm begins to overwrite the existing snapshots with the newer ones (e.g., snapshot-1000 is overwritten with snapshot-6000 and so on). As can be seen in Table 3.2, each training iteration was initialized with the weight from the previous training, except for training iteration zero, which was initialized with the snapshot from the model zoo, as mentioned above, and training iteration four part one, which was initialized with the snapshot from training iteration two. Table 3.2 also lists the maximum internal iterations performed for each training, after which each training was finished.

For each training, the calculated losses and their learning rates were displayed for each 500th internal iteration and stored in a log file, which can be found in the corresponding training folder. For the learning rates, the multi-step function was used, where DLC automatically updates the learning rates during training according to the settings [34]. These settings are also presented in Table 3.2.

Training iteration	Weights initialized	Learning rates	Max internal iterations	Number of frames
0	Model zoo "full dog" – snapshot-75000	0.005 until 2000 0.02 until 5000 0.002 until 15000	15000	480
1	Iteration 0 – snapshot-15000	0.005 until 7000 0.002 until 10000	10000	480
2	Iteration 1 – snapshot-10000	0.02 until 8000 0.002 until 11000 0.001 until 15000	15000	640
3	Iteration 2 – snapshot-15000	0.001 until 5000	5000	640
4 - part 1	Iteration 2 – snapshot-15000	0.005 until 5000 0.002 until 7000 0.001 until 10000	10000	800
4 - part 2	Iteration 4 (part 1) – snapshot-10000	0.02 until 5000 0.002 until 8000 0.001 until 10000	10000	800
5	Iteration 4 (part 2) – snapshot-10000	0.002 until 7000 0.001 until 15000	15000	1160

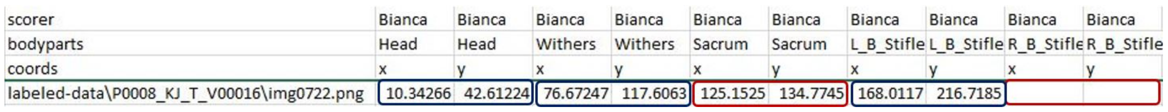
Table 3.2: Settings used for the training iterations. Each training iteration started with the last stored weights from the previous one, except for training iteration four, where the first part was initialized with the weights from number two, and training iteration zero, which was initialized with the weights from the model zoo. For the learning rates, multiple steps were used for the internal iterations, so that DLC automatically changes the learning rate during training according to the settings. For each training iteration, a maximum of internal iterations was set. The last column represents the amount of data available for the training.

After training iteration four – part two, a network refinement was performed. Three of the videos used for the training were randomly selected (video number 16, 19, and 21) and analyzed by the model, i.e., all predictions were made by the trained model. From these analyzed videos, 40 frames were extracted using the outlier algorithm "jump". This algorithm extracts frames where the predictions of body parts differ from the predictions in the previous frame by more than a certain threshold epsilon [34]. This epsilon describes the distance at which the body parts leap between the frames [34] and was set to 20 px.



scorer	DLC_resnet50_AIDogJul8shuffle1_10000														
bodyparts	Head			Withers			Sacrum			L_B_Stifle			R_B_Stifle		
coords	x	y	likelihood	x	y	likelihood	x	y	likelihood	x	y	likelihood	x	y	likelihood
721	11.05396	42.22662	0.804816	76.78342	117.7687	0.969513	130.6556	132.8153	0.049515	168.3845	216.3572	0.98104	122.4732	244.241	0.273086
722	10.34266	42.61224	0.80511	76.67247	117.6063	0.969067	2.903493	44.2542	0.050738	168.0117	216.7185	0.980216	19.15022	259.4416	0.27695

(a) Comparison frame 0721 and 0722



scorer	Bianca	Bianca	Bianca	Bianca	Bianca	Bianca	Bianca	Bianca	Bianca	Bianca
bodyparts	Head	Head	Withers	Withers	Sacrum	Sacrum	L_B_Stifle	L_B_Stifle	R_B_Stifle	R_B_Stifle
coords	x	y	x	y	x	y	x	y	x	y
labeled-data\P0008_KJ_T_V00016\img0722.png	10.34266	42.61224	76.67247	117.6063	125.1525	134.7745	168.0117	216.7185		

(b) Corrected coordinates for frame 0722

Figure 3.4: Outlier extraction example:

Figure 3.4 shows an example of how this algorithm works, using an excerpt from the analysis of video number 16. As mentioned earlier, the algorithm "jump" compares two consecutive frames at a time, e.g., frames 721 and 722, as illustrated in the upper picture (a) in Figure 3.4. The blue rectangles highlight the x and y coordinates that do not have a large gap, whereas the red rectangles mark coordinates whose difference is larger than the defined epsilon threshold indicating a wrong prediction. In this example the x and y coordinates of the sacrum and the x coordinate of the right back stifle were affected. Because of these discrepancies, frame number 722 is identified as an outlier. This process is repeated until all outlier frames are found. Usually this is a huge number of frames for longer videos like in this case. However, since some of these are similar to each other and thus would have a similar learning effect, it is not necessary to use all of the frames. Therefore, only 40 frames of each video were extracted, using the already described "kmean" parameter to ensure that these frames are as different as possible. Via the graphical use interface, the wrong labeled body parts were corrected by hand and the edited frames were added to the training data set. Picture (b) of Figure 3.4 shows the frame number 722 as part of the training data set. As can be seen in the blue rectangle, the labels for the head, withers and left back stifle did not change since these predictions were made properly and therefore no refinement was necessary. The x and y coordinates for the sacrum (left red rectangle) were adjusted and the label for the right back stifle (right red rectangle) was completely deleted, since it was not visible at all in this frame. After correcting all outlier frames, the training was continued with training iteration five. With this process it was possible to improve the performance of the algorithm without extending the data set.

For the evaluation after each training iteration, DLC calculates the mean average Euclidean error (MAE)[4] between the manually generated labels and those predicted by the algorithm in px. During each evaluation process, a total of four MAE values are calculated. These include the training error with and the training error without p-cutoff, as well as the test error with and the test error without p-cutoff. The p-cutoff variable is used to identify body part predictions with low likelihood values by serving as a specified threshold. This indicates that while all predictions are included in the calculations of the two MAEs "without p-cutoff", only predictions that fully satisfy the requirement

[4] Note: The MAE is proportional to the average root mean square error (RMSE).

likelihood > p-cutoff were considered in the calculations “with p-cutoff”. This distinction allows a better understanding of the trained model’s performance (see Section 4.1.1). Note: For the test errors, frames from the test set are used to evaluate the performance of the algorithm on unseen data, i.e., data not used for training. [34]

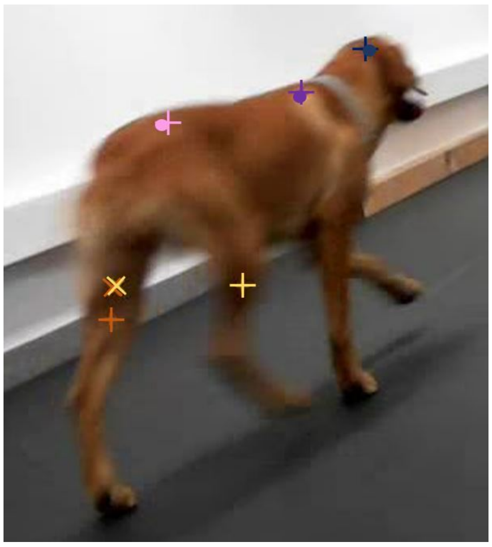


Figure 3.5: Example of a test frame after the evaluation of training iteration one: The plus symbol represents the manually labeled body parts. The DLC predictions for the head, sacrum, and withers were predicted with a likelihood greater than the p-cutoff threshold (set to 60%) and are therefore marked with a dot. For the stifles, the predictions are labeled with an “x”, indicating a $likelihood \leq p\text{-}cutoff$.

Additionally, the body parts with a likelihood smaller than the p-cutoff are highlighted differently in the evaluation frames. Figure 3.5 shows an example of a test frame that was used for the evaluation of training iteration one. The plus symbol indicates the manually labeled body parts, the predictions of DLC are labeled either as a dot, if the predictions were made with a likelihood greater than the p-cutoff threshold, or as “x” in the event of a likelihood smaller than the p-cutoff [34]. The p-cutoff threshold was set to 60% for each evaluation process. As is visible in Figure 3.5, the predictions for the head, withers, and sacrum fully meet this requirement (marked with dot), but the two stifles were predicted with a likelihood of less than 60% (marked with “x”).

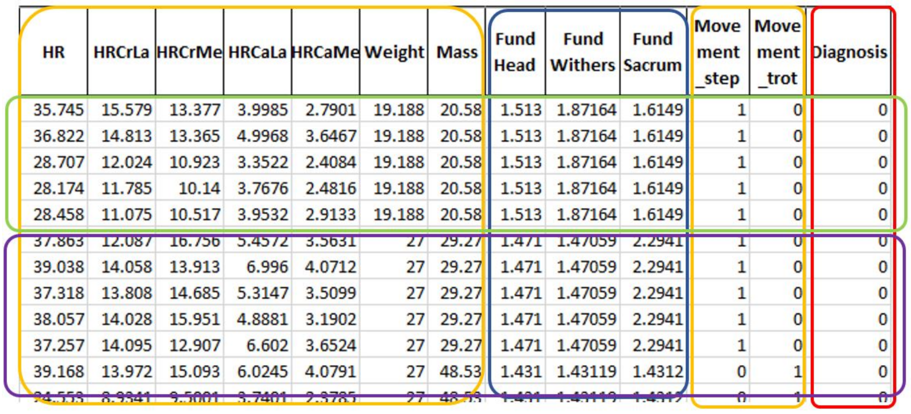
With training iteration five, the training process was stopped. The model’s performance on unseen data was tested with patients number six and seven, whose data were not part of the training data set. Since its performance was sufficient (see results and discussion in Section 4.1.2), this model was used for the body part prediction in the application, i.e., all available gait analysis videos were analyzed with this model. The output files of these analyzes, which contain the x and y coordinates as well as the corresponding likelihood values of each body part per frame, were stored as Excel files. These data were appropriately processed to enable the extraction of a total of three features (explained in Section “Feature extraction from the DLC file” in subchapter 3.3.1: Data preparation), which, along with the features extracted from the provided pressure data, were used as input for the XGBoost classifier.

3.3. Classification with XGBoost

This section describes the data preparation procedure for the XGBoost classifier for the provided PA data and the body part predictions obtained from the DLC part. Furthermore, the performed parameter study is explained and the testing of the trained classifier is outlined.

3.3.1. Data preparation

During data preparation, a data frame is created whose rows contain the patient data and whose columns represent the features that XGBoost uses for decision making. The selection of the features is a very important step, as the accuracy of the classifier depends on it (see Section 3.3.2: Parameter study). The final data frame consists of 24 features extracted from the PA data, three features extracted from the body part prediction files, and, for training purposes, one target column. Figure 3.6 shows an excerpt from this data frame covering eleven features and the target column. The features for the hind right (HR) paw including its four areas craniolateral (CrLa), craniomedial (CrMe), caudolateral (CaLa) and caudomedial (CaMe), the movement, the mass, and the dog's weight are extracted from the PA files (see orange rectangles). The blue rectangle marks the three features extracted from the DLC files, while the red rectangle highlights the target column containing the diagnoses. The green and purple rectangles indicate the rows belonging to two different patients. The individual steps of the data preparation are described below.



HR	HRCrLa	HRCrMe	HRCaLa	HRCaMe	Weight	Mass	Fund Head	Fund Withers	Fund Sacrum	Movement_step	Movement_trot	Diagnosis
35.745	15.579	13.377	3.9985	2.7901	19.188	20.58	1.513	1.87164	1.6149	1	0	0
36.822	14.813	13.365	4.9968	3.6467	19.188	20.58	1.513	1.87164	1.6149	1	0	0
28.707	12.024	10.923	3.3522	2.4084	19.188	20.58	1.513	1.87164	1.6149	1	0	0
28.174	11.785	10.14	3.7676	2.4816	19.188	20.58	1.513	1.87164	1.6149	1	0	0
28.458	11.075	10.517	3.9532	2.9133	19.188	20.58	1.513	1.87164	1.6149	1	0	0
37.863	12.087	16.756	5.4572	3.5631	27	29.27	1.471	1.47059	2.2941	1	0	0
39.038	14.058	13.913	6.996	4.0712	27	29.27	1.471	1.47059	2.2941	1	0	0
37.318	13.808	14.685	5.3147	3.5099	27	29.27	1.471	1.47059	2.2941	1	0	0
38.057	14.028	15.951	4.8881	3.1902	27	29.27	1.471	1.47059	2.2941	1	0	0
37.257	14.095	12.907	6.602	3.6524	27	29.27	1.471	1.47059	2.2941	1	0	0
39.168	13.972	15.093	6.0245	4.0791	27	48.53	1.431	1.43119	1.4312	0	1	0
34.553	8.9341	9.5001	3.7401	2.3785	27	48.53	1.431	1.43119	1.4312	0	1	0

Figure 3.6: Excerpt from the data frame used for parameter study and training showing some of the columns representing the features used for decision-making (orange and blue rectangles) and the target column, which is only used for training purposes (red rectangle). The rows contain the values of the patients extracted from the files, with the green rectangle highlighting the lines that belong to one patient and the purple rectangle representing the ones that already belong to the next patient. This data frame is used for the parameter study as well as for the training and consists of 27 features, one target column, and 104 rows.

Note: Unless otherwise noted, the examples and data extracts used below refer to patient number one, analyzed during the step/walk and diagnosed as healthy (P0001_H_S_V0001).

Feature extraction from the PA files

The features selected from the provided PA Excel files are the pressure data (forces) for each paw and its four areas CrLa, CrMe, CaLa, and CaMe, the calculated mass, and the weight of the dog. In addition, the "movement" feature is added to distinguish between the two gaits studied, the trot and step/walk. To recall, each paw is divided into four sections: CrLa, CrMe, CaLa, and CaMe. Caudal refers to the opposite direction, that is, towards the tail, while cranial represents the direction towards the head. Medial relates to the "middle of the dog," whereas lateral corresponds to the directions to the sides of the dog. An abstract image of these areas for the left paws (forepaw and hind paw areas are identical) is shown in Figure 3.7. For the right paws, lateral and medial are reversed, ie. CrMe and CaMe are on the left side of the right paw and CrLa and CaLa are on the right side. The forces acting on each of these areas during a stance phase are measured with the pressure plate and summed using the PA.

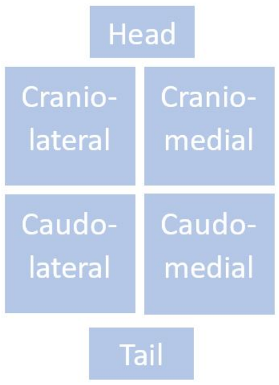
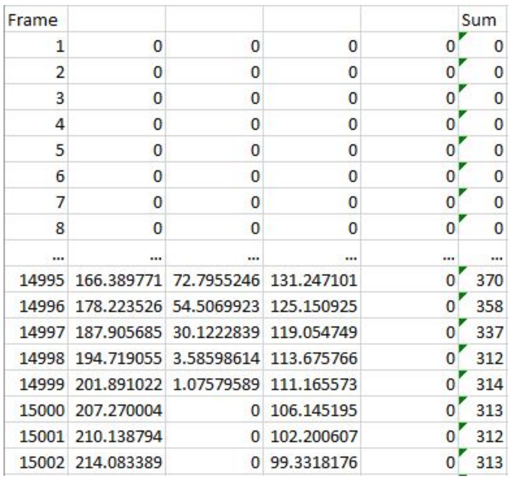


Figure 3.7: Abstract illustration of the left paw's areas

Since XGBoost does not allow non-numeric columns [29], one-hot encoding, a process by which categorical features are converted into a form that can be provided to ML algorithms, was used. During this process, the feature of each sample corresponding to its original category is assigned the value of one. Thus, the "movement" feature is split into the two new columns "Movement_step" and "Movement_trot", specifying one for all rows where a "step" or a "trot" was present and zero for the remaining rows (see Figure 3.6 orange rectangle on the right). For the training process, it is necessary to add a target column, which contains the diagnoses made by the physicians, at the end of the data frame. To obtain a numerical column here as well, zero is set if the dog is diagnosed as healthy and one if the dog is lame.

In order to utilize the measured forces as input for the XGBoost, the data for each paw and each of its areas is averaged separately. To extend the data set, these averages are calculated for each pass instead of using the entire measured data at once. A pass, as already mentioned in Section 1.1.2, represents the time period in which the dog passes the pressure plate exactly once. For example, patient one shows measured data for a total of five passes, resulting in five rows in the data frame instead of just one. It should be noted that the PA tool did not store data for ALL passes done during the gait analyses, which means that the number of passes performed by the dog does not match the number of passes used for classification.

The frame numbers of the video and the corresponding measurement data for the paws are listed on the sheet "Data Frame" in each Excel file. Figure 3.8 illustrates an excerpt from this sheet, where column one lists the frame numbers and columns two through five contain the measured forces for the FL, front right (FR), hind left (HL), and HR paws (in that order). In frame number 14995, for instance, the measured value for the FL is 166.389771, while the HR is reported as zero, indicating that this leg was not on the pressure plate at that time.



Frame					Sum
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
...	...	...	...	...	...
14995	166.389771	72.7955246	131.247101	0	370
14996	178.223526	54.5069923	125.150925	0	358
14997	187.905685	30.1222839	119.054749	0	337
14998	194.719055	3.58598614	113.675766	0	312
14999	201.891022	1.07579589	111.165573	0	314
15000	207.270004	0	106.145195	0	313
15001	210.138794	0	102.200607	0	312
15002	214.083389	0	99.3318176	0	313

Figure 3.8: "Data Frame" sheet excerpt from the provided Excel file: Column one shows the frame numbers of the video, columns two to five contain the measured forces, whereby a zero indicates that the paw did not touch the measure plate. The last column sums up the forces measured in each frame.

From this "Data Frame" sheet, the numbers of the start and end points for each pass are extracted and used to calculate the duration of the passes with Equation 3.1. With the help of these computed values, the averages for each paw and each pass are calculated according to Equation 3.2 and added to the data frame, where each pass is stored in a new row. Thus, the four features FL, FR, HL, and HR are extracted, containing the averaged forces of the paws.

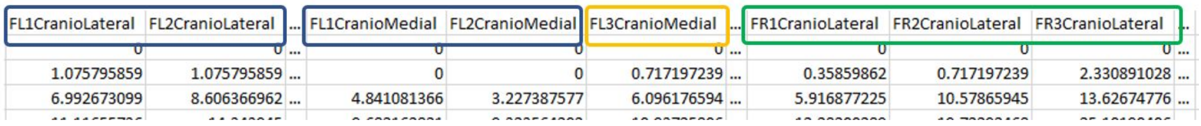
$$pass\_duration = end\_point\_pass - start\_point\_pass \tag{3.1}$$

$$paw\_avg\_pass = \frac{\sum_{start\_point\_pass}^{end\_point\_pass} pressure\_data\_pass}{pass\_duration} \tag{3.2}$$

The pressure data measured for the four areas of each paw are stored separately in the "Data Quadrant Area" sheet of the Excel files and are therefore not listed next to the frame number where they were measured[5]. To ensure that the correct values are averaged and appended to the data frame, it is necessary to know how many steps were measured during the complete gait analysis and how many steps belong to the individual passes.

For example, for patient number one, there is measurement data for ten steps of the FL paw and twelve steps of the FR paw, and as already mentioned, this patient has a total of five passes. By determining the number of steps per pass, it was found that on the first pass, the FL paw touched the pressure plate twice while the FR paw touched it three times. Thus, for the calculation of the average forces of each paw area, the first **two** columns of the FL paw areas and the first **three** columns of the FR paw areas must be considered.

[5] Note: The first column is called "Frame", but it does not correspond to the frames in the videos. It only shows a consecutive numbering of the lines.



FL1CranioLateral	FL2CranioLateral	...	FL1CranioMedial	FL2CranioMedial	FL3CranioMedial	...	FR1CranioLateral	FR2CranioLateral	FR3CranioLateral	...
0	0	...	0	0	0	...	0	0	0	...
1.075795859	1.075795859	...	0	0	0.717197239	...	0.35859862	0.717197239	2.330891028	...
6.992673099	8.606366962	...	4.841081366	3.227387577	6.096176594	...	5.916877225	10.57865945	13.62674776	...
[illegible]	[illegible]		[illegible]	[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	

Figure 3.9: "Data Quadrant Area" sheet excerpt from the provided Excel file: The picture shows the first two steps for the CrLa and the CrMe areas of the FL paw (blue rectangles), which are part of the first pass for patient number one. The yellow rectangle highlights the third step of the CrMe area of the FL paw, which is already part of the next pass. For the FR paw, three steps have touched the pressure plate during the first pass, so the first three columns are used to calculate the average for this first pass (the green rectangle highlights this for the CrLa area).

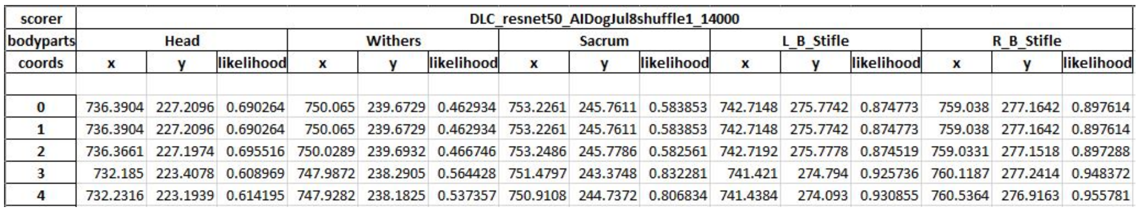
Figure 3.9 shows an excerpt of the "Data Quadrant Area" sheet, highlighting the columns for calculating the average force of the first pass for the CrLa and CrMe areas of the FL paw in blue rectangles and the columns used for the CrLa area of the FR paw in the green rectangle. The yellow rectangle shows the CrMe area from the third step of the FL paw, which is already part of the calculation for the second pass. Finally, 16 additional features are extracted, namely the averaged forces for the four areas CrLa, CrMe, CaLa, and CaMe of each paw. The following equation was used to calculate these averages:

$$area\_avg\_pass = \frac{\sum\limits_{number\_steps\_per\_pass} area\_pressure\_data\_pass}{pass\_duration} \tag{3.3}$$

To distinguish between big and small dogs, the mass as well as the weight of each patient are added as features. Both values are stored in the Excel file sheet "Übersicht". One of the great advantages of models based on decision trees is that they are not affected by the scale of the input, so no normalization of the data is required [37]. With that, the feature extraction from the data provided by the PA tool is finished. To recap, four features represent the averaged forces for each paw, 16 features depict the averaged forces for each of the four areas per paw, one feature represents the mass, one feature indicates the dog's weight, and two features represent the distinction between the step/walk and trot.

Feature extraction from the DLC prediction files

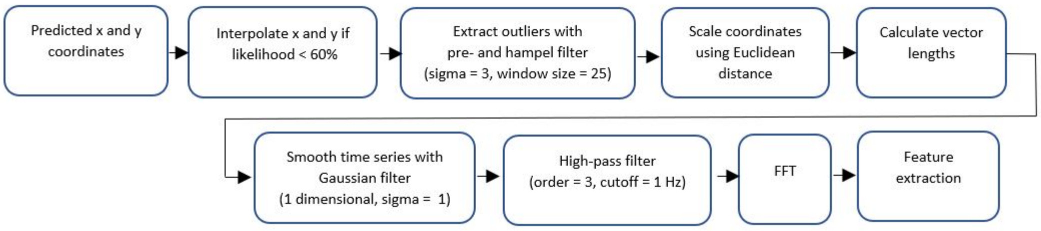
The output files from the DLC algorithm contain the predicted x and y coordinates (in px) as well as the likelihood value for each body part per frame. Figure 3.10 depicts an excerpt of one of these output files, illustrating the scorer, i.e., the used model for analyzing the videos, the body parts and the corresponding predictions. The first column refers to the frames of the video, e.g., at frame zero, the model predicted the dog's head at x = 736.3904 px and y = 227.2096 px with a likelihood of 69.0264%.



scorer	DLC_resnet50_AIDogJul8shuffle1_14000														
bodyparts	Head			Withers			Sacrum			L_B_Stifle			R_B_Stifle		
coords	x	y	likelihood	x	y	likelihood	x	y	likelihood	x	y	likelihood	x	y	likelihood
0	736.3904	227.2096	0.690264	750.065	239.6729	0.462934	753.2261	245.7611	0.583853	742.7148	275.7742	0.874773	759.038	277.1642	0.897614
1	736.3904	227.2096	0.690264	750.065	239.6729	0.462934	753.2261	245.7611	0.583853	742.7148	275.7742	0.874773	759.038	277.1642	0.897614
2	736.3661	227.1974	0.695516	750.0289	239.6932	0.466746	753.2486	245.7786	0.582561	742.7192	275.7778	0.874519	759.0331	277.1518	0.897288
3	732.185	223.4078	0.608969	747.9872	238.2905	0.564428	751.4797	243.3748	0.832281	741.421	274.794	0.925736	760.1187	277.2414	0.948372
4	732.2316	223.1939	0.614195	747.9282	238.1825	0.537357	750.9108	244.7372	0.806834	741.4384	274.093	0.930855	760.5364	276.9163	0.955781

Figure 3.10: Excerpt from a prediction file showing the predicted x and y coordinates (in px) as well as the likelihood values for each body part and frame. In the first line, the model used for analysis is noted as scorer.

The fundamental frequencies are extracted from the movement of the head, withers, and sacrum and used as features for the classifier. These features are added to the data frame accordingly as "FundHead", "FundWithers", and "FundSacrum". The steps of the signal processing from the available coordinates to the extraction of the frequencies are outlined in Figure 3.11 and are explained below using the head movement of patient number ten during trot (P0010_KJ_T_V0020), unless otherwise noted.



Predicted x and y coordinates
Interpolate x and y if likelihood < 60%
Extract outliers with pre- and hampel filter (sigma = 3, window size = 25)
Scale coordinates using Euclidean distance
Calculate vector lengths
Smooth time series with Gaussian filter (1 dimensional, sigma = 1)
High-pass filter (order = 3, cutoff = 1 Hz)
FFT
Feature extraction

Figure 3.11: Signal processing for feature extraction from the DLC prediction files

In a first step, all x and y coordinates are interpolated for which the likelihood value is below the specified threshold of 0.6 (60%). This value for the threshold was chosen because a prediction with a likelihood of more than 60% is already considered sufficient. Since this interpolation can only be performed if there is data before and after the area to be corrected, affected rows at the beginning and the end of the data cannot be interpolated and are therefore deleted. This can be seen, for example, in Figure 3.10, where the probability for the withers within the first five frames and for the sacrum within the first three frames is below this threshold, i.e., the first five frames are deleted for this patient.

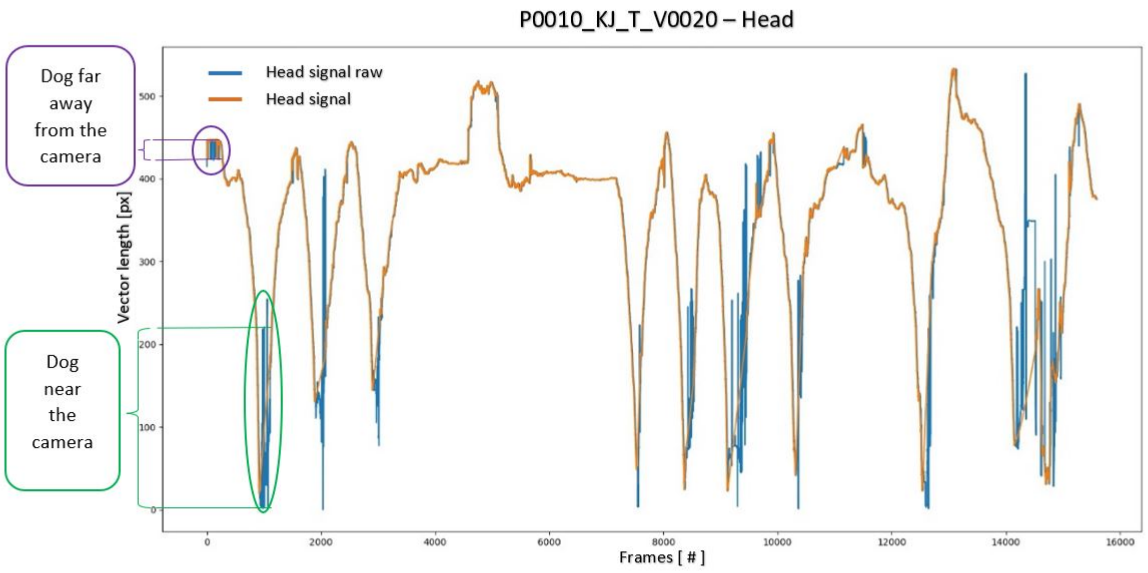


Figure 3.12: Raw signal vs. filtered signal of head marker for patient number ten. The orange signal indicates the head movement after the pre-filter and Hampel filter, while the blue signal corresponds to the raw (unfiltered) data of the head. The purple circle marks an area where the dog is in the back of the room, far away from the camera, while the green circle represents an area where the patient is close to it.

In the next step, outliers are extracted using a pre-filter followed by a Hampel filter. Figure 3.12 shows the head signals of patient number ten before (blue signal) and after (orange signal) filtering, clearly demonstrating that outliers have been removed from the raw data. The pre-filter calculates the absolute z-values for each of the data points (x and y coordinates), relative to the sample mean μ and the standard deviation σ (see Equation 3.4). Each of these calculated values is compared to a specified threshold, with the values for the x and y coordinates replaced by a "NaN" if either or both z-values are below that limit. This threshold is set to 3σ (three times the standard deviation) according to the so-called "3-σ-rule", a simple and popular heuristic for detecting outliers [38]. The outliers determined in this way are finally interpolated linearly, with the number of successive interpolation points limited to 20.

$$|z| = \frac{data - \mu}{\sigma} \tag{3.4}$$

The Hampel filter, one of the most robust and efficient outlier identifier, slides a moving window over the entire data. On each window a robust adjustment of the "3-σ-rule" named Hampel identifier is applied to characterise each data point with respect to its preceding and subsequent samples. [39] Thus, the filter calculates the median and estimates the standard deviation sigma, using the median absolute deviation (MAD) ($\sigma \approx 1.4826\text{MAD}$) for each window. After that, each data point within a window is compared to its calculated mean, and if the value deviates by more than three times the median absolute deviation, it is treated as an outlier and replaced by the median value. For the size of the sliding windows, a value of 25 is defined[6].

Since the camera position was not ideal during the experiments, the dog walks, as already mentioned, more or less obliquely towards and away from the camera. Consequently, the patient appears larger in the frames where it is close to the camera and smaller in the frames where it is far away. The resulting issue is that the dog's movements are clearly visible near the camera but the further away he gets from the camera, the smaller and harder they are to see. Additionally, this phenomena also happens with outliers, i.e., when the dog is near the camera, an outlier results in a considerably larger discrepancy between the x and y coordinates of the preceding frame and those of the "outlier-frame". For instance, the length of the outlier jags visible in Figure 3.12 (raw signal) are significantly larger in the green circle ($\approx$ 210px), where the patient is near the camera, than those in the purple circle ($\approx$ 20px), where the dog is far away. Depending on where the dog is located in the video, a scaling of the x and y coordinates is applied as a workaround. For this purpose, first the Euclidean distance (see Equation 3.5) between the sacrum and the hind stifle in each frame is calculated, using the right stifle when the dog moves away from the camera and the left stifle when it moves towards the camera. This distinction is made because the left back stifle is more visible than the right back stifle when the dog is walking towards the camera and vice versa.

$$eucl\_dist = \sqrt{((x\_stifle - x\_sacrum)^2 + (y\_stifle - y\_sacrum)^2)} \tag{3.5}$$

After calculating the Euclidean distances, the scaling factors are computed by taking the first distance as the reference (its scaling factor is set to one) and using the equation below to get the remaining scaling factors for each frame:

$$scaling\_factor = \frac{ref\_distance}{eucl\_dist} \tag{3.6}$$

[6] Note: Several window sizes were tried, 25 was the best fit.

Finally, the x and y coordinates for each frame of the head, withers, and sacrum are multiplied by the appropriate scale factor (see Equations 3.7 and 3.8) to ensure that the movements are independent of the dog's position in the video and thus comparable.

$$x\_scaled = x * scaling\_factor \tag{3.7}$$

$$y\_scaled = y * scaling\_factor \tag{3.8}$$

The position vectors' lengths are computed in the following step using the scaled coordinates. Since the algorithm has its coordinate origin in the upper left corner, the vectors calculated for points near the left edge of the frame are short and become larger as the distance of the dog to the left edge increases. As a result, there are peaks in the signal that coincide with the dog backing away from and moving towards the camera.

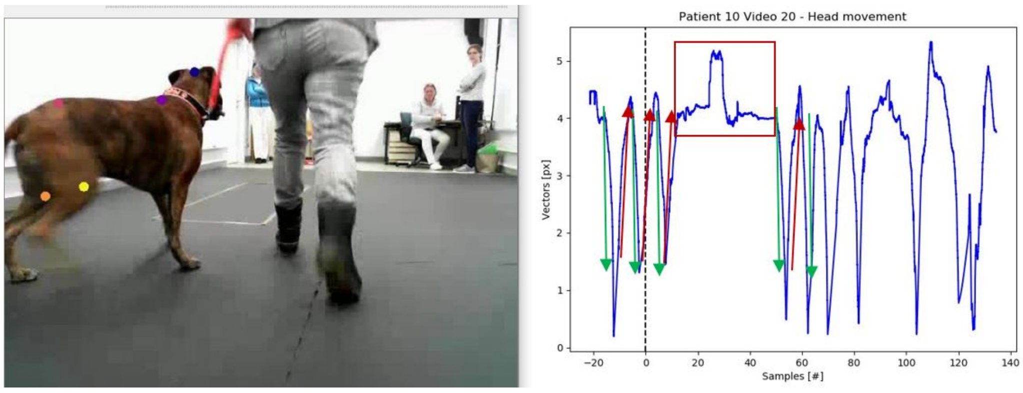


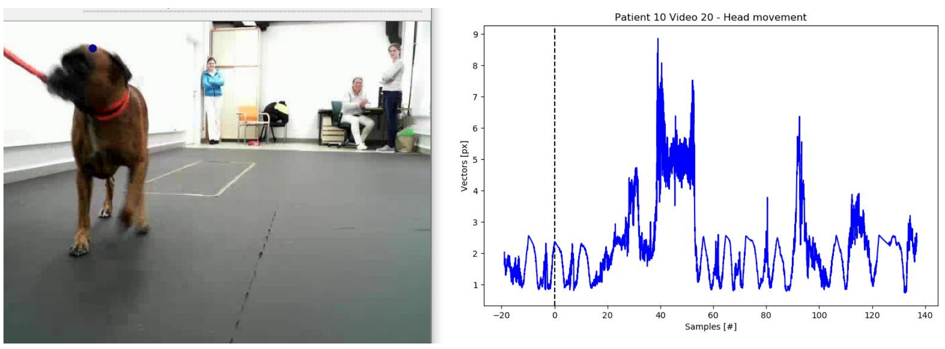
Figure 3.13: Screenshot of the labeled video and the corresponding head movement: The labeled video was played synchronously with the movement signal of the head of patient number ten. The green arrows indicate the dog's movement towards the camera, decreasing the distance to the coordinate origin of the algorithm, and the red arrows refer to the dog's movement away from the camera (increasing distance). The red rectangle marks the area where the dog was only in the back of the room.

This behavior is illustrated in Figure 3.13 for the head signal of patient number ten, where the green arrows indicate movement towards the camera and the red arrows indicate movement in the opposite direction. In other words, each peak to the top corresponds to the position at which the dog is farthest away. The red rectangle highlights an area in the signal where the patient did not move forward, but remained in the background for a while. To counteract this interference, the coordinate origin used for the calculation of the vectors is shifted from the upper left corner to the upper middle of the dog's walking range. Therefore, the maximum x coordinate is identified and divided by two to get x_half. Including this value in the calculation of the vectors, as shown in Equation 3.9, results in a shift of the y-axis . This is done separately for all frames, ensuring that in each frame the distances from the shifted y-axis to the furthest point of the dog to the left and to the right are equal.

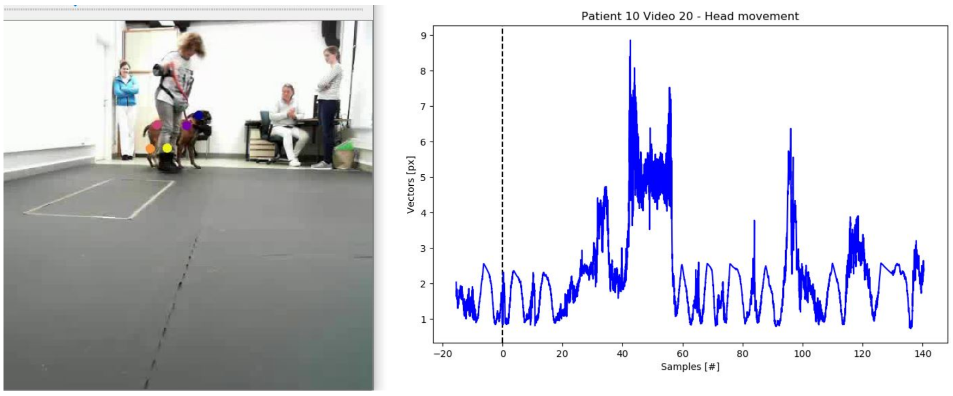
$$vector\_length = \sqrt{(x - x\_half)^2 + y^2} \tag{3.9}$$

After this process step, the time series is smoothed with a Gaussian filter, using a sigma of one. Figure 3.14 shows two snapshots from the gait analysis of patient ten including the signal after the scaling, shifting, and smoothing process. The video is played synchronously with the data once again, with the dashed line indicating the current point in the signal. Both images were taken in such a way that a peak in the signal is shown. In the top picture (a), the dog is very close to the camera, but in the bottom picture (b), it is far away, indicating that the peaks are now independent of the dog's position and merely illustrate the movements of the head.

Since a fundamental frequency of more than 1 Hz is expected, the signal is high-pass filtered (order = 3, cutoff = 1 Hz) before it is finally transformed to the frequency domain using the fast Fourier transform (FFT). The frequency with the highest intensity in each case is extracted as the fundamental frequency of the head, neck, and sacrum. Thus, the three features from the DLC files are extracted and added to the data frame. As these features are extracted on the basis of the entire gait analysis and not on the basis of the individual passes, as is the case with the features of the PA data, the extracted frequencies are the same for each of the rows belonging to one patient.



(a) Dog near the camera



(b) Dog further away from the camera

Figure 3.14: Screenshot of the labeled video and corrected head signal: The dashed lines represent the corresponding point in the signal for the currently displayed image. Picture (a) shows the snapshot when the dog is near the camera, while the patient is far away in image (b). For both cases, there is now a peak at an upward position in the signal (marked by the dashed lines), which means that the peaks are independent of the dog's position in the video.

3.3.2. Parameter study

The data frame used for the parameter study consists of 27 features (columns), the target column (diagnosis), and 104 rows containing the processed data of patients number two to eleven, as already mentioned in Section 3.3.1. In the file P0002_H_T_V004, the data for the paw area is missing for one of the steps, i.e., for this step made within the gait analysis, there is force data for the paw but no force data for its areas CrLa, CrMe, CaLa, and CaMe. Since it is not possible to identify the step at which this occurred, the file cannot be processed during data preparation and is therefore not part of the final data frame. Patients number one and twelve are not included in either the training or the parameter study process, as they act as test patients to validate the trained XGBoost.

For the parameter study, different combinations of the features were used as input to train the XGBoost Classifier, where the features "Mass", "Weight", and "Movement"[7] were present in each combination to distinguish between large and small dogs as well as between the trot and step/walk. The combinations are listed in Table 3.3, where the "Forces" represents the 20 features containing the averaged forces of the paws and their four areas. For each combination, training and test performance, which takes values between zero (poor) and one (perfect), as well as the accuracy value (in percentage) were calculated and displayed. Additionally, the log losses for the training data set and the receiver operating characteristics (ROC) curves including the area under curve (AUC) values are illustrated in a diagram for all combinations. The train-test split was set to 80:20, meaning that 80% of the data set was used for training, with the random state parameter set to five to ensure that this split was always the same for each run of the script and each combination, so the results are comparable.

	Mass	Weight	Movement	Forces	FundHead	FundWithers	FundSacrum
1	x	x	x		x		
2	x	x	x			x	
3	x	x	x				x
4	x	x	x		x	x	
5	x	x	x		x		x
6	x	x	x			x	x
7	x	x	x		x	x	x
8	x	x	x	x			
9	x	x	x	x	x		
10	x	x	x	x		x	
11	x	x	x	x			x
12	x	x	x	x	x	x	
13	x	x	x	x	x		x
14	x	x	x	x		x	x
15	x	x	x	x	x	x	x

Table 3.3: Feature combinations used in the parameter study.

Before the parameter study starts, the "RandomizedSearchCV" (randomized search with cross-validation) function provided by sklearn is used to find the most suitable hyperparameters for the classifier with respect to the input data. This way, the function forms all possible combinations of the hyperparameters defined in a grid and outputs the combination that yields the best accuracy [29]. As fine-tuning all XGBoost hyperparameters would be very time-consuming, the focus is on

[7] Note: The two separate columns for step and trot are simply summarized as "Movement" in the text for the sake of simplicity.

the seven hyperparameters most commonly fine-tuned by machine learning experts [29], which are briefly described in the following sections.

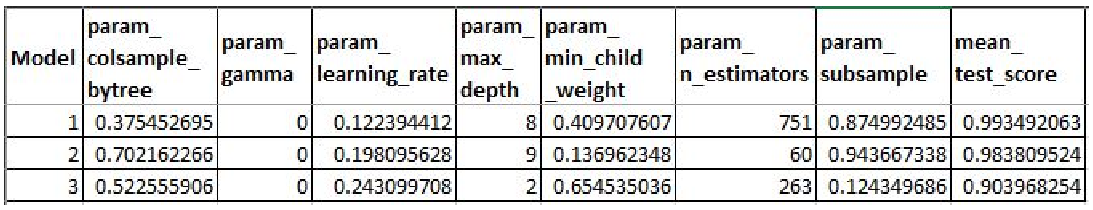
N_estimators defines how many trees are trained on the residuals and is per default set to 100. Increasing this value could lead to an improvement of the score by using a large data set. For each boosting round, the defined value of the **learning_rate**, whose default value is 0.3, decreases the weights of the trees. Reducing this value prevents overfitting on the one hand (since the transferred weights are smaller) but also means that more trees are needed to achieve good accuracy. Another hyperparameter used to prevent overfitting is the **max_depth**, which limits the length of the tree. **Gamma**, also known as the Lagrange multiplier, sets a bar that nodes must clear in order to split further with respect to the loss function. Its default value is zero, and although there is no upper limit for the gamma, a value higher than ten is already assumed to be excessive. [29]

The minimum sum of weights required within a node to be split further is defined by the hyperparameter **min_child_weight**, and by means of **subsample**, the threshold for the percentage of training instances (rows) for each boosting round is specified. Setting the subsample below 100% and increasing the min_child_weight reduces overfitting. The last hyperparameter **colsample_bytree** is used to reduce the impact of the features by randomly picking specific columns according to the given percentage. [29]

The hyperparameter values used as input for the "RandomizedSearchCV" function are listed in Table 3.4. For n_estimators, max_depth, and gamma, only integer values from a specified range were used, ensuring that the default values are included. The remaining parameters were defined to cover float values between zero and one. The search was done using the repeated stratified k-fold with five splits and three repetitions, resulting in 15 cross-validations per iteration. A total of 500 iterations were performed to find the best fitting model, using "accuracy score" for the evaluation of each hyperparameter combination and split. The results of the randomized search are saved in the "Results_RandomizedSearch.xlsx" file and can be found in the "Software\Classification\Param_Study" folder.

Hyperparameter	Start value	End value	Value type
N_estimators	50	800	Integer
Learning_rate	0	1	Float
Max_depth	1	10	Integer
Gamma	0	1	Integer
Min_child_weight	0	1	Float
Subsample	0	1	Float
Colsample_bytree	0	1	Float

Table 3.4: Feature combinations used in the parameter study.

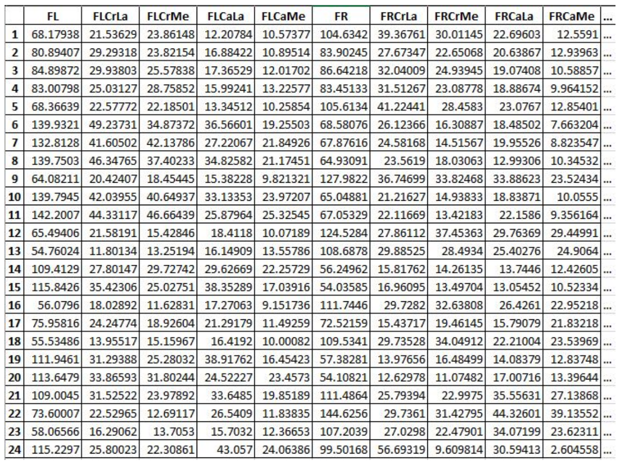


Model	param_ colsample_ bytree	param_ gamma	param_ learning_rate	param_ max_ depth	param_ min_child _weight	param_ n_estimators	param_ subsample	mean_ test_score
1	0.375452695	0	0.122394412	8	0.409707607	751	0.874992485	0.993492063
2	0.702162266	0	0.198095628	9	0.136962348	60	0.943667338	0.983809524
3	0.522555906	0	0.243099708	2	0.654535036	263	0.124349686	0.903968254

Figure 3.15: Result extract of the randomized search summarizing only the three selected models of interest for the parameter study. The model numbering has been added by hand and displays a sequential numbering of the three selected models not indicating any ranking.

Reviewing the results, the best scored model achieved a mean test score of 0.99349206, as displayed in the first row of Figure 3.15 (model one). Since this model would require 751 estimators, leading to a high computational cost, another model with a similar score but fewer estimators was chosen. This model's hyperparameters along with its score, which is 0.98380952, are depicted in the second row of Figure 3.15. In this model (hereafter referred to as model two), the accuracy of almost all feature combinations was 100%, indicating a problem with the hyperparameters (the results for this model can be found in the text file "ParamStudy_Model2"). Looking at the calculated performances of all combinations, the train performances were all rated 1.0, which could be an indication of overfitting. An additional hint that model two may be overfitting is the fact that the test scores for eleven of the 15 splits computed during the randomized search also have a value of 1.0. Thus, model two is not used for the parameter study, and when selecting the model for the parameter study, care was taken to select one whose test values were all below 1.0. Model number three shown in Figure 3.15 fulfills these criteria and achieved a good mean test score of 0.903968254.

For the final training of the XGBoost classifier, all 27 features were used as input, and the train and test log losses as well as the accuracy were calculated and plotted for evaluation purposes. Since the hyperparameters, the train-test split, and the random state were the same as for the training during the parameter study, the accuracy is equal to that of the parameter study, where all features were used. The trained model was finally tested on unseen data from patients one and twelve. An excerpt of this test data frame is shown in Figure 3.16 and consists of the 27 features, the target column for evaluation purposes and 24 rows, containing only the data from the files P0001_H_S_V0001, P0001_H_T_V0002, P0012_KJ_S_V0021, and P0012_KJ_T_V0022.

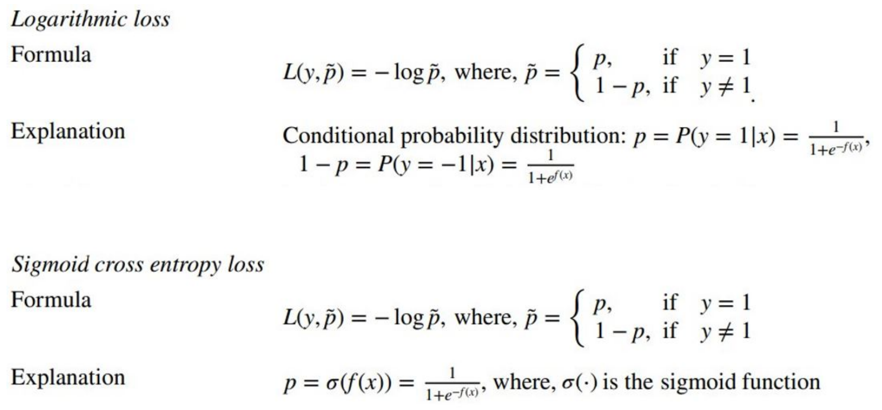


	FL	FLCrLa	FLCrMe	FLCaLa	FLCaMe	FR	FRCrLa	FRCrMe	FRCaLa	FRCaMe	...
1	68.17938	21.53629	23.86148	12.20784	10.57377	104.6342	39.36761	30.01145	22.69603	12.5591	...
2	80.89407	29.29318	23.82154	16.88422	10.89514	83.90245	27.67347	22.65068	20.63867	12.93963	...
3	84.89872	29.93803	25.57838	17.36529	12.01702	86.64218	32.04009	24.93945	19.07408	10.58857	...
4	83.00798	25.03127	28.75852	15.99241	13.22577	83.45133	31.51267	23.08778	18.88674	9.964152	...
5	68.36639	22.57772	22.18501	13.34512	10.25854	105.6134	41.22441	28.4583	23.0767	12.85401	...
6	139.9321	49.23731	34.87372	36.56601	19.25503	68.58076	26.12366	16.30887	18.48502	7.663204	...
7	132.8128	41.60502	42.13786	27.22067	21.84926	67.87616	24.58168	14.51567	19.95526	8.823547	...
8	139.7503	46.34765	37.40233	34.82582	21.17451	64.93091	23.5619	18.03063	12.99306	10.34532	...
9	64.08211	20.42407	18.45445	15.38228	9.821321	127.9822	36.74699	33.82468	33.88623	23.52434	...
10	139.7945	42.03955	40.64937	33.13353	23.97207	65.04881	21.21627	14.93833	18.83871	10.0555	...
11	142.2007	44.33117	46.66439	25.87964	25.32545	67.05329	22.11669	13.42183	22.1586	9.356164	...
12	65.49406	21.58191	15.42846	18.4118	10.07189	124.5284	27.86112	37.45363	29.76369	29.44991	...
13	54.76024	11.80134	13.25194	16.14909	13.55786	108.6878	29.88525	28.4934	25.40276	24.9064	...
14	109.4129	27.80147	29.72742	29.62669	22.25729	56.24962	15.81762	14.26135	13.7446	12.42605	...
15	115.8426	35.42306	25.02751	38.35289	17.03916	54.03585	16.96095	13.49704	13.05452	10.52334	...
16	56.0796	18.02892	11.62831	17.27063	9.151736	111.7446	29.7282	32.63808	26.4261	22.95218	...
17	75.95816	24.24774	18.92604	21.29179	11.49259	72.52159	15.43717	19.46145	15.79079	21.83218	...
18	55.53486	13.95517	15.15967	16.4192	10.00082	109.5341	29.73528	34.04912	22.21004	23.53969	...
19	111.9461	31.29388	25.28032	38.91762	16.45423	57.38281	13.97656	16.48499	14.08379	12.83748	...
20	113.6479	33.86593	31.80244	24.52227	23.4573	54.10821	12.62978	11.07482	17.00716	13.39644	...
21	109.0045	31.52522	23.97892	33.6485	19.85189	111.4864	25.79394	22.9975	35.55631	27.13868	...
22	73.60007	22.52965	12.69117	26.5409	11.83835	144.6256	29.7361	31.42795	44.32601	39.13552	...
23	58.06566	16.29062	13.7053	15.7032	12.36653	107.2039	27.0298	22.47901	34.07199	23.62311	...
24	115.2297	25.80023	22.30861	43.057	24.06386	99.50168	56.69319	9.609814	30.59413	2.604558	...

Figure 3.16: Excerpt from the test data frame, which only contains the data from the two test patients one and twelve, resulting in 24 rows. In the picture, ten of the 26 extracted features are visible.

3.4. Loss functions and evaluation techniques used

As can be seen in Figure 3.17, the **sigmoid cross-entropy loss function** used to evaluate the DLC algorithm and the **logarithmic loss (log loss)** used for the XGBoost evaluation, are identical. These identical losses are characterized by two titles since their definitions come from different sources. While the cross-entropy loss derives the probability value from the predicted value using the sigmoid activation function, the log loss acquires the probability value via the conditional probability distribution (see logarithmic loss explanation in Figure 3.17). In this distribution, *p* stands for the likelihood of being predicted as a member of a positive class, while *1-p* stands for the opposite probability. The logarithmic loss rises as the likelihood of a correct prediction goes down, and it decreases when the probability of a correct prediction is close to one. The gap between the actual and predicted outputs is referred to as cross entropy. The two probability distributions are more closely spaced apart the lower the cross entropy value. As a result, sigmoid cross entropy loss and logarithmic loss have the same meaning. [40]



Logarithmic loss	
Formula	$L(y, \tilde{p}) = -\log \tilde{p}$, where, $\tilde{p} = \begin{cases} p, & \text{if} \quad y = 1 \\ 1 - p, & \text{if} \quad y \neq 1 \end{cases}$.
Explanation	Conditional probability distribution: $p = P(y = 1 \mid x) = \frac{1}{1+e^{-f(x)}}$, $1 - p = P(y = -1 \mid x) = \frac{1}{1+e^{f(x)}}$
Sigmoid cross entropy loss	
Formula	$L(y, \tilde{p}) = -\log \tilde{p}$, where, $\tilde{p} = \begin{cases} p, & \text{if} \quad y = 1 \\ 1 - p, & \text{if} \quad y \neq 1 \end{cases}$
Explanation	$p = \sigma(f(x)) = \frac{1}{1+e^{-f(x)}}$, where, $\sigma(\cdot)$ is the sigmoid function

Figure 3.17: Formula and explanation of log loss and sigmoid cross-entropy loss.

The **ROC** is an evaluation metric used in binary classification problems that represents a probability curve for several decision thresholds. Based on the threshold, the classifier categorizes each of the instances. For example, with a threshold of 0.5, all instances with a prediction probability above 0.5 will be classified as positive, all others as negative. The terminology of positive and negative class is related to the sigmoid function, which outputs a probability value between zero and one: positive if the prediction probability of the instance is within the area between the threshold and one; negative if it's within the region between the threshold and zero. The ROC curve is plotted with true positive rate (TPR) (see Equation 3.10), also known as sensitivity, on the y-axis and the false positive rate (FPR), which results from one minus the specificity (see Equation 3.12), on the x-axis. The sensitivity calculates the proportion of the positive class, which got correctly classified, while the level of specificity (see Equation 3.11) reveals how much of the negative class was correctly categorized (true negative rate).

$$TPR = \frac{true\ positive}{true\ positiv + false\ negative} \tag{3.10}$$

$$Specificity = \frac{true\ negative}{true\ negative + false\ positive} \tag{3.11}$$

$$FPR = 1 - Specificity = \frac{false\ positive}{true\ negative + false\ positive} \tag{3.12}$$

The area under the ROC curve is represented by the **AUC** value, which shows the ability of the trained model to differentiate between the positive and negative class. The more accurately the model predicts the classes, the higher is this value. The ideal model would have an AUC value of one, which means that it optimally differentiates between the classes, i.e., all positive instances are classified as positive and all negative instances are classified as negative; there are no false predictions. This is reflected in the left picture of Figure 3.18, where the value for the FPR is zero and the TPR value is one. In the second image from the left, the worst case scenario can be seen: an AUC value of approximately 0.5 means that the model is not able to distinguish between the positive and negative classes, in other words, the model is deciding randomly for each instance. Any AUC value less than 0.5 means that the amount of incorrectly predicted instances is greater than the amount of the correctly predicted ones. Thus, a value of zero means that all positive instances are classified as negative and all negative instances are classified as positive, resulting in a FPR value of one and a TPR value of zero (see picture to the right). The third image from the left displays the ROC curve for an AUC value of 0.7, indicating that the model can reliably distinguish between positive and negative classes with a 70% chance. Each AUC value between 0.5 and one means that the model identifies more true positives and true negatives than false positives and false negatives.

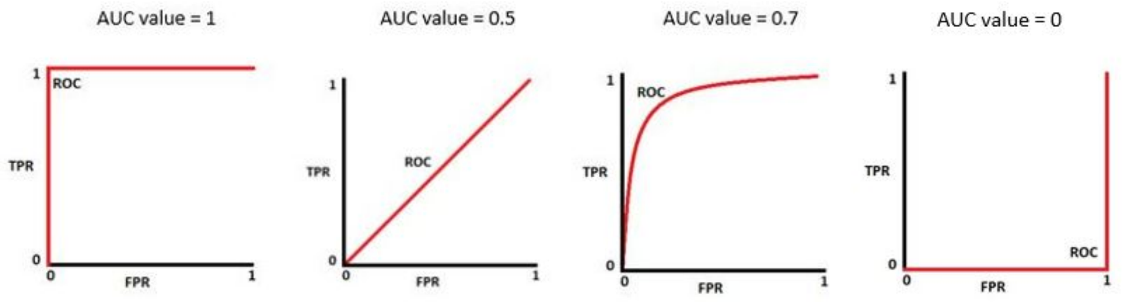


Figure 3.18: ROC curve and AUC value examples.

4. Results and discussion

4.1. DLC body part prediction results

In this section, the results of the training iterations, including the evaluation results and the cross-entropy loss, as well as the results of the predictions of the final model for unseen data are presented and discussed.

4.1.1. Evaluation of training iterations

As briefly described in Section 3.2.2, the training process of the model included a total of seven training iterations. After each of these iterations, the model was evaluated with test data (see Table 4.1 for results) and the cross-entropy loss was plotted over the internal iterations. Using the evaluation results and the plot, the learning progress of the model was interpreted after each training iteration, and depending on the outcome, adjustments were made before starting the next training iteration. These adaptations included either changes in the learning rates, adding training data, and/or extracting outliers. The results, plots, and adjustments made for each training iteration are outlined in the following sections.

The following table summarizes the evaluation results of each training iteration.

Training iteration	Internal iterations	Train error (px)	Test error (px)	Train error with p-cutoff (px)	Test error with p-cutoff (px)	p-cutoff
0	7000	19.26	28.02	5.29	8.27	0.6
1	10000	14.39	19.42	6.64	8.61	0.6
2	15000	7.4	15.69	4.57	5.80	0.6
3	3000	6.40	20.44	4.73	7.48	0.6
4	20000	9.56	11.03	5.14	5.22	0.6
5	13000	9.07	8.03	5.00	5.46	0.6
	14000	8.55	7.98	4.76	5.45	0.6
	15000	9.40	8.89	5.25	8.68	0.6

Table 4.1: Evaluation results of training iterations showing the test and train errors (with and without p-cutoff), the number of maximum internal iterations, and the used p-cutoff threshold for each training iteration.

Training iteration 0

As can be seen in Table 4.1, this model, which has been trained with 7000 internal iterations, has relatively high train and test errors (19.26 px and 28.02 px). However, if only the predictions with a probability greater than 60% are considered, the values are already in a good range (5.29 px and 8.27 px). Although only 480 frames were used for the first training iteration, excellent initial results were achieved thanks to transfer learning. By initializing the model with the weights from the pre-trained model zoo algorithm, the model did not have to learn from scratch and thus only needed 7000 internal iterations to reach train and test errors with p-cutoff < 9 px.

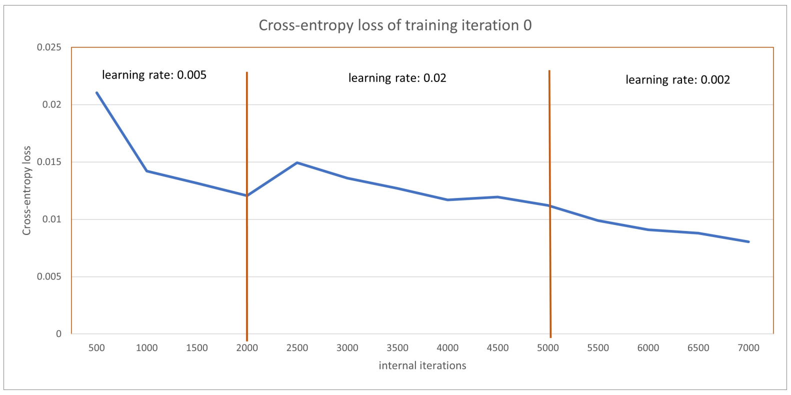


Figure 4.1: Cross-entropy loss diagram for training iteration zero: At the beginning a learning rate of 0.005 is used, which is automatically updated to 0.02 after the 2000th internal iteration. Starting with internal iteration number 5000, the learning rate is updated again and a value of 0.002 is used.

The cross-entropy loss plot for training iteration zero is shown in Figure 4.1, with the vertical lines designating the internal iterations where the learning rate was adjusted. Note: The smaller the learning rate, the finer the updates of the weights. The learning rate values used are recommended by DLC and have therefore been retained as such. Variations were only made during the updates. Initially, a learning rate of 0.005 was used, which was changed to 0.02 at the 2000th internal iteration. As can be observed in Figure 4.1, the cross-entropy loss decreases quite rapidly at the beginning, and as the learning rate changes, it starts to increase for some internal iterations, reaching a peak at internal iteration 2500. This behavior is normal, as the weights are now updated with a larger value. After this peak, the cross-entropy loss steadily decreases, with this decrease slowing down as expected with the update of the learning rate to 0.002.

Since the timing of updating learning rates always depends on the application and the amount of data available for the training, there is no basic rule for it. Therefore, especially for the first training, it was tested more or less randomly how the loss evolves for 480 existing frames at different learning rates.

Training iteration 1

This training iteration is the continuation of training iteration zero, i.e., the last saved snapshot of the previous training was used to initialize the weights. The training and test errors without the p-cutoff values decreased further and are now 14.39 px and 19.42 px, while the errors with p-cutoff values increased slightly to 6.64 px and 8.61 px (see Table 4.1). This increase could be due to the fact that DLC predicts a larger number of body parts with a probability above 60% and/or the distances for these predictions to the ground truth labels are still quite large.

Figure 4.2 illustrates the cross-entropy loss for this training iteration. Since the learning rate of 0.002 in training iteration zero decreased very slowly, this training iteration was initialized with a slightly higher learning rate of 0.005 and only updated to 0.002 with the 7000th internal iteration. In the beginning, the cross-entropy loss decreased relatively fast, but afterwards the learning progress was very slow, indicating that the selected learning rates were too small [18]. Therefore, this training iteration was terminated with the 15000th internal iteration and continued as training iteration two with different learning rate settings.

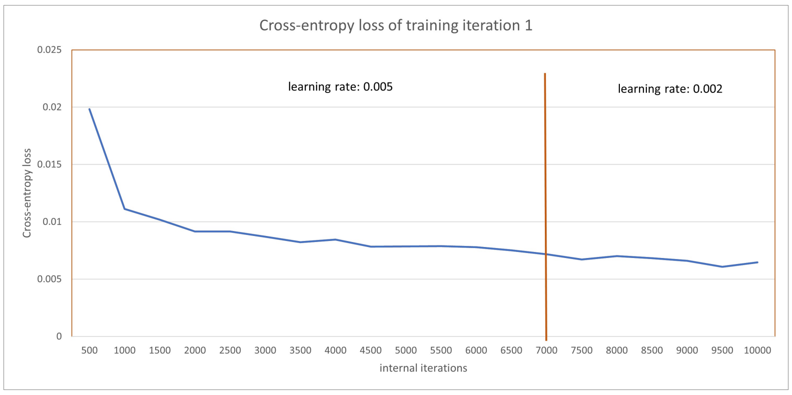


Figure 4.2: Cross-entropy loss diagram for training iteration one: The learning rate is initialized with 0.005 and updated after the 7000th internal iterations to 0.002.

Training iteration 2

Training iteration two was initialized with snapshot 10000 from training iteration one. The errors for the train and test evaluation have further decreased for both the calculations with and without the p-cutoff threshold. Although the error rate is decreasing, it can be seen in Table 4.1 that the gap between the train and test errors is increasing (4.57 px vs. 5.80 px with p-cutoff and 7.4 px vs. 15.69 px without p-cutoff).

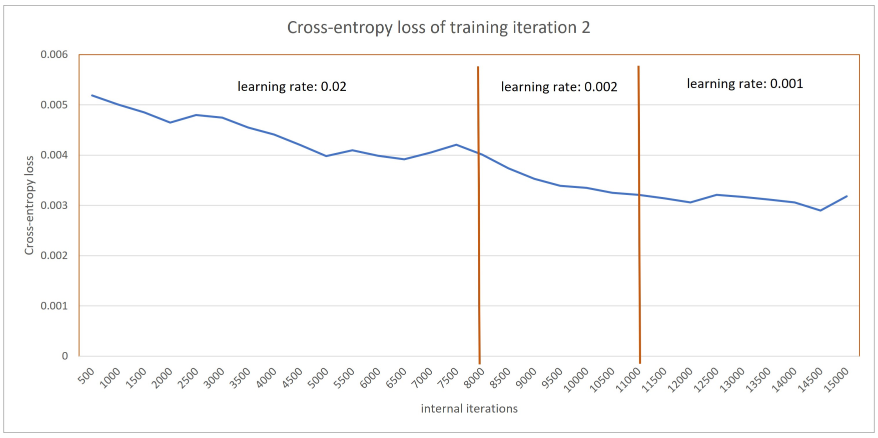


Figure 4.3: Cross-entropy loss diagram for training iteration two: At the beginning, the learning rate was set to 0.02, which was updated to 0.002 at internal iteration 8000. With iteration 11000, the learning rate was updated to 0.001.

The diagram of the cross-entropy loss for this training iteration (see Figure 4.3) shows very nicely the effect of changing the learning rate to a higher value. Due to the relatively small learning rates chosen for the previous training, it was not possible to achieve a loss below 0.005 even after 10000 internal iterations (see Figure 4.2). However, with the learning rate of 0.02 used for training iteration two, the loss exceeded 0.005 relatively quickly, as shown in Figure 4.3. In the range of 8000 to 11000 internal iterations, a learning rate of 0.002 was used, which still resulted in a good reduction of the loss. For the last few internal iterations, the learning rate was updated to 0.001 to make smaller steps towards the ground truth data, since the loss was already close to 0.003. As a result, the cross-entropy loss first reaches a brief plateau and then suddenly begins to increase again. This behavior, along with the increased gaps between the two train and test errors, indicates a possible incipient overfitting of the model.

Training iteration 3

To determine whether the sudden increase in cross-entropy loss from the previous training iteration indicated overfitting, training iteration two was continued into training iteration three at the same learning rate (0.001). A separate training iteration was chosen so that the weights of the model prior to overfitting would remain stored separately and not be overwritten by further internal iterations, which would be unusable when overfitting occurs.

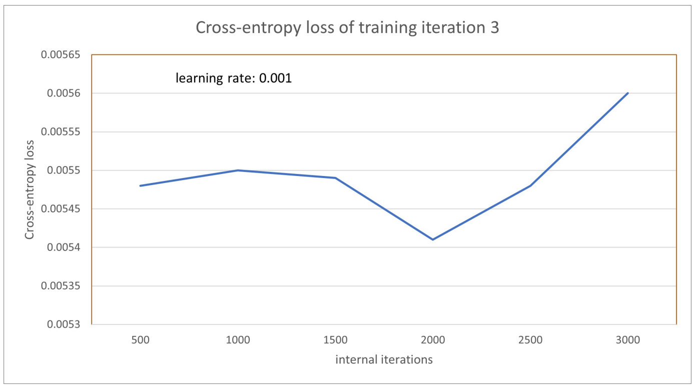


Figure 4.4: Cross-entropy loss diagram for training iteration three: With a learning rate of 0.001, the progress was quite slow and stable until the loss decreased rapidly before it started to increase.

The evaluation of this training iteration shows an even larger gap between train and test error without p-cutoff (6.4 px vs. 20.44 px) and an increase in the train error with p-cutoff of 0.16 px and for the test error with p-cutoff 1.68 px. Together with the fact that, as can be seen in Figure 4.4, the loss only increases from iteration 2000, it was proven that overfitting occurs with continued training. Therefore, training iteration three was excluded and training iteration four was initialized with the weights of training iteration two after extending the training data set.

Training iteration 4

As can be seen in Table 4.1, the train errors slightly increased compared to the evaluation results of training iteration two (as mentioned above, training iteration three was excluded). On the one hand, the training error without p-cutoff increased from 7.4 px to 9.56 px, and the training error with p-cutoff increased from 4.57 px to 5.14 px. The test errors, on the other hand, decreased to 11.03 px (without p-cutoff) and 5.22 px (with p-cutoff). These are exactly the results that were expected after adding new data to the training data set. The increase in the training error indicates that more training data is available and that the algorithm needs to be trained more with this expanded data set. The reduced errors in the test evaluation, on the other hand, show that the model performs more effectively on unseen data as a result of additional training data.

Training iteration four was divided into two parts, since the training for the first part was interrupted after 10000 internal iterations to adjust the learning rates. Figure 4.5 illustrates the cross-entropy loss for both parts in one diagram. For the first part, the learning rates were chosen to be quite small. With a learning rate of 0.005 until the 5000th internal iteration, the loss was in the range of 0.0085 and 0.007. However, after the learning rate was first changed to 0.002 and then to 0.001 (at iteration number 7000), the learning progress was again quite slow. Therefore, the training was stopped at this point to change the learning rate to the higher value of 0.02.

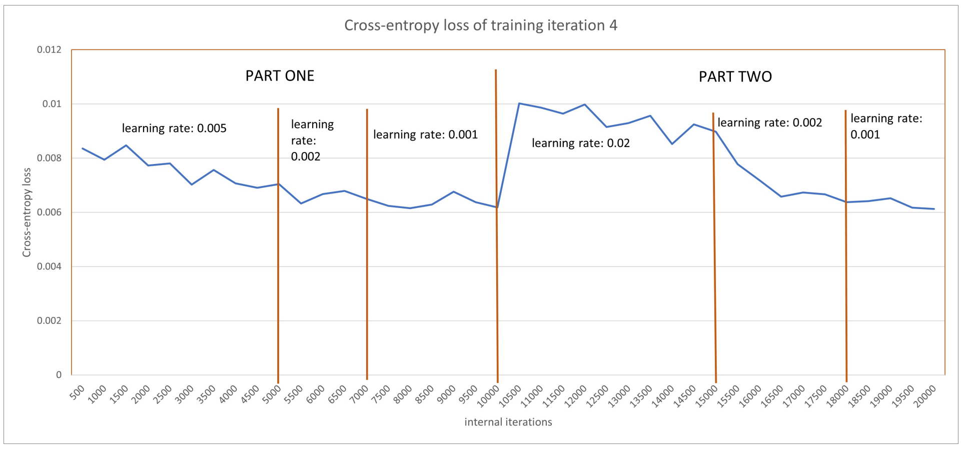


Figure 4.5: Cross-entropy loss diagram for training iteration four: After 10000 internal iterations (part one), the training was interrupted to adjust the learning rates. Afterwards, the training was continued until 20000 internal iterations (part two).

With this change, the cross-entropy loss increased as expected with the initial internal iterations, but then started to decrease in a positive way. Even after changing the learning rate to 0.002, the decrease was still satisfactory. At internal iteration 18000, the learning rate value was updated again, with the loss settling just above 0.006. With the already low learning rate, the cross-entropy loss did not decrease further, so the data set used for training had to be adjusted to achieve a better result. For this purpose, a network refinement was performed before the next training iteration was started.

Training iteration 5

For this iteration, the learning rate was initially set to 0.002 in order to make slightly "larger" steps in the beginning and was changed to 0.001 at the 7000th internal iteration. As can be seen in Figure 4.6, the learning progress was good in the first half of the training but started to flatten out in the second half. Towards the end of the training, the loss began to alternately increase and decrease, indicating the beginning of overfitting.

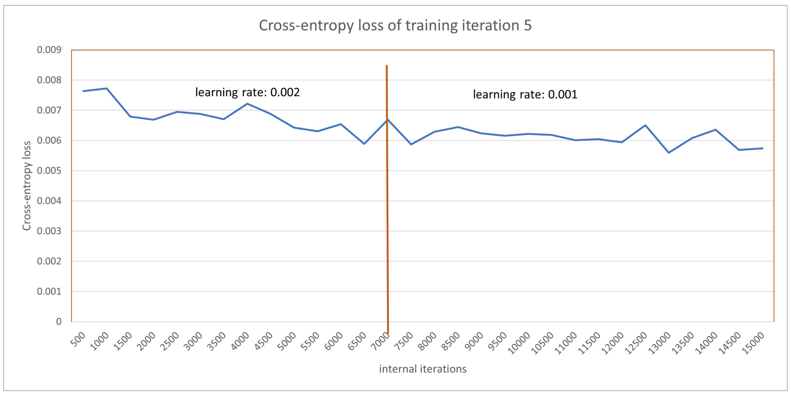


Figure 4.6: Cross-entropy loss diagram for training iteration five: The initial learning rate of 0.002 was changed to 0.001 in the middle of the training, resulting in a slight increase of the cross-entropy loss. In the second half of the training, the loss was quite stable until it began to alternate at the last few thousand internal iterations.

The evaluation results for the last saved snapshot (15000) indicate a train error of 9.40 px and a test error of 8.89 px, i.e., compared to training iteration four, there is a reduction of the errors without p-cutoff. Both error results with p-cutoff increased, with the training error rising from 5.14 px to

5.25 px and the test error growing from 5.22 px to 8.68 px. The fact that this test error is smaller than the training error could be explained by the possibility that more of the “easier” cases ended up in the test data set when the data was split into test and training sets. Since the errors increased with p-cutoff and the diagram of the cross-entropy loss showed a strange behavior at the end, two more evaluations were performed with the same model using the snapshots 13000 and 14000.

As can be seen in Table 4.1, iteration 13000 shows a similar behavior to snapshot 15000. The train error and the test error without p-cutoff decreased to 9.07 px and 8.03 px, with the test error being smaller than the train error once again. For the values with p-cutoff, the train error was reduced to 5.00 px, but the test error increased to 5.46 px. For snapshot 14000, the results for the train and test errors without p-cutoff are 8.55 px and 7.98 px, respectively. So it is still the case that the train error is larger, but now the distance between these two errors is only 0.57 px instead of 1.04 px, as it is the case for snapshot 13000. The train error with p-cutoff went down to 4.76 px and the test error increased slightly to 5.45 px (see Table 4.1).

Final DLC model

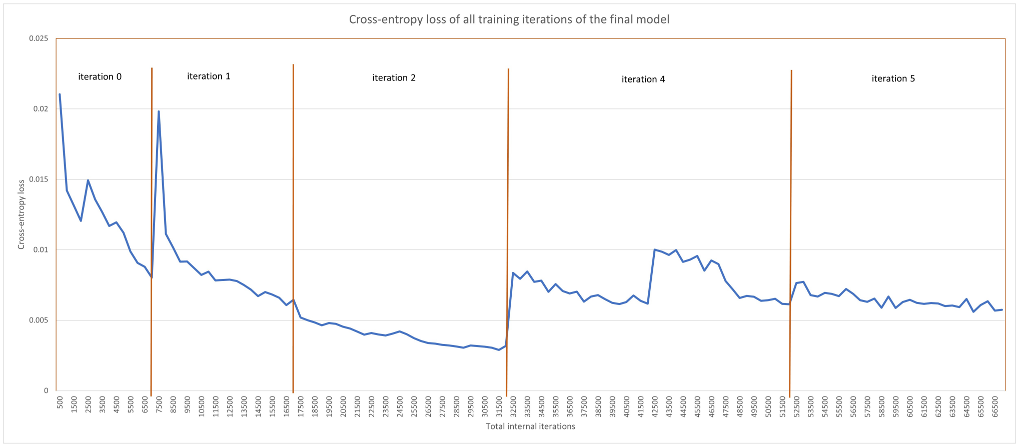


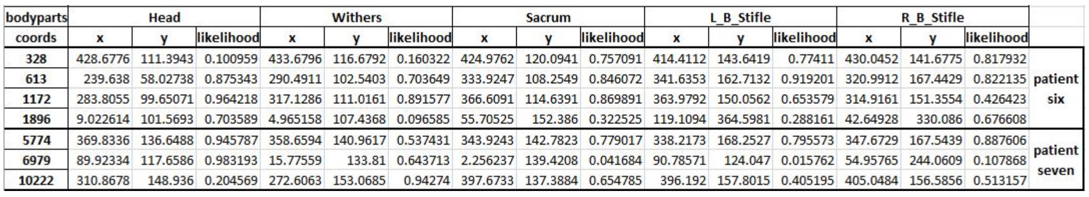
Figure 4.7: Cross-entropy loss diagram for all training iterations

In Figure 4.7, the cross-entropy loss for all training iterations (except the excluded training iteration three) is plotted in one graph. Although the loss was smallest after training iteration two, the training was continued with additional data because this model did not perform well on the unseen (test) data. This is reflected in the relatively large gap between the train and test errors (without p-cutoff) in Table 4.1. Since the learning progress started to flatten out during training iteration five and there was no more data to add, the training process was terminated after a total of 67000 internal iterations. With respect to the evaluation results of the last three snapshots from training iteration five described above, it was decided to use snapshot 14000 from iteration five as the final model.

To recap, the final model used for the body part prediction of the videos was trained with 1160 frames extracted from 20 different videos belonging to ten different patients and took 66000 iterations (snapshot 14000). This model achieved a cross-entropy loss of 0.00636 and train and test errors of 9.40 px and 8.89 px (without p-cutoff) and 4.76 px and 5.45 px (with p-cutoff).

4.1.2. Predictions on unseen data

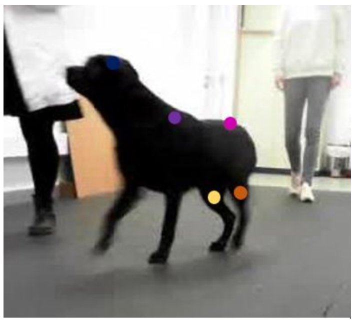
In order to check whether the performance of the final model is sufficient, labeled videos were created for patients six and seven and manually reviewed. As already mentioned above, these two patients were excluded from the training process, meaning that the algorithm has not seen these videos during training. For the creation of labeled videos, DLC provides a function to save all labeled frames of the video in a corresponding folder. All four labeled videos including their separately stored frames as well as the prediction files for the test patients are located in the folder "Software\Classification\Test_patients".



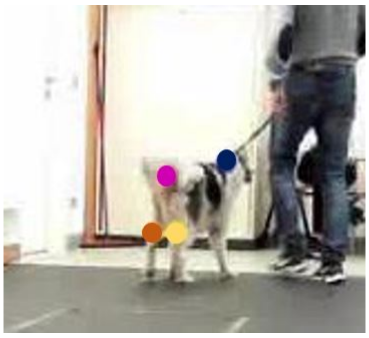
bodyparts	Head			Withers			Sacrum			L_B_Stifle			R_B_Stifle			
coords	x	y	likelihood	x	y	likelihood	x	y	likelihood	x	y	likelihood	x	y	likelihood	
328	428.6776	111.3943	0.100959	433.6796	116.6792	0.160322	424.9762	120.0941	0.757091	414.4112	143.6419	0.77411	430.0452	141.6775	0.817932	
613	239.638	58.02738	0.875343	290.4911	102.5403	0.703649	333.9247	108.2549	0.846072	341.6353	162.7132	0.919201	320.9912	167.4429	0.822135	**patient six**
1172	283.8055	99.65071	0.964218	317.1286	111.0161	0.891577	366.6091	114.6391	0.869891	363.9792	150.0562	0.653579	314.9161	151.3554	0.426423	
1896	9.022614	101.5693	0.703589	4.965158	107.4368	0.096585	55.70525	152.386	0.322525	119.1094	364.5981	0.288161	42.64928	330.086	0.676608	
5774	369.8336	136.6488	0.945787	358.6594	140.9617	0.537431	343.9243	142.7823	0.779017	338.2173	168.2527	0.795573	347.6729	167.5439	0.887606	
6979	89.92334	117.6586	0.983193	15.77559	133.81	0.643713	2.256237	139.4208	0.041684	90.78571	124.047	0.015762	54.95765	244.0609	0.107868	**patient seven**
10222	310.8678	148.936	0.204569	272.6063	153.0685	0.94274	397.6733	137.3884	0.654785	396.192	157.8015	0.405195	405.0484	156.5856	0.513157	

Figure 4.8: Predicted coordinates and likelihood values for selected frames of patient number six and patient number seven.

To illustrate some important aspects of the algorithm's performance, the following sections use selected frames from the folders "temp-P0006_H_T_V0012" (patient six, video twelve) and "temp-P0007_KJ_S_V0013" (patient seven, video 13) as well as the likelihood values from the associated prediction files P0006_H_T_V0012.xlsx and P0007_KJ_S_V0013.xlsx. Since the stored labeled frames from the videos had their frame numbers included in the file names and these numbers are also reflected in the Excel files, each prediction (coordinates and likelihood) can be explicitly assigned to the associated frame. Figure 4.8 summarizes the predicted coordinates as well as the likelihood values for the seven frames used as examples in the next sections. For the marking of the body parts, DLC uses the following color scheme: **head – blue, withers – purple, sacrum – pink, left back stifle – orange, and right back stifle – yellow**. Body parts predicted with a $likelihood \leq 0.6$ are not marked in the frames.



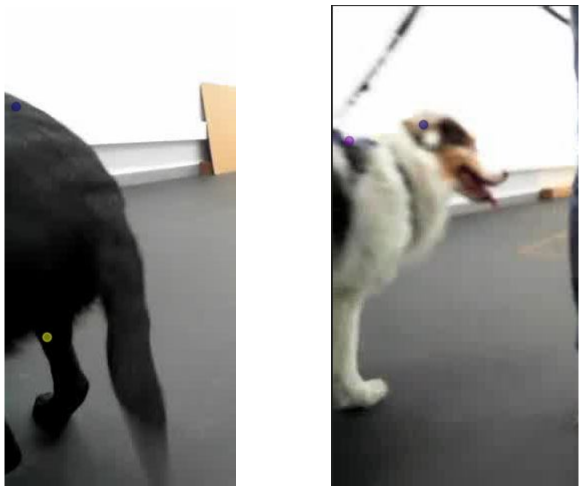
(a) Patient six – frame 00613



(b) Patient seven – frame 05774

Figure 4.9: Labeled frames with good predictions: For patient number six (a) all body parts were predicted correctly with likelihood values bigger than 60%. The case is similar for patient number seven, where only the withers is not marked in the picture because it is predicted with a likelihood smaller than the threshold.

As visible in the left image (a) of Figure 4.9, the algorithm successfully identified all five body parts for patient number six. For patient number seven (image (b)), head, sacrum as well as the left and right back stifles are labeled properly, only the withers is not marked in the picture. However, considering that the withers is predicted with a likelihood of 53.74%, as illustrated in Figure 4.8 for frame number 5774, which is just barely below the specified threshold, it is still a good performance.



(a) Patient six – frame 01896

(b) Patient seven – frame 06979

Figure 4.10: Labeled frames with moderate predictions: The left image (a) shows the back of patient number six, where the model correctly predicted the right back stifle (yellow dot), but completely mis predicted the head (blue dot). For the anterior part of patient number seven, which is shown in the right image (b), both the head and the withers were predicted properly.

Since the dogs turn around outside the camera field before moving away from the camera again, each video contains some frames in which only parts of the dogs can be seen. Two corresponding examples are shown in Figure 4.10, where only the posterior part of patient six in the left image and only the anterior part of patient seven in the right image visible. Since DLC does not output "NaN" values, non-visible body parts are also predicted, but with very small likelihood values in well-trained models. For this reason, an interpolation of the x and y coordinates of body parts whose likelihood values are below the specified threshold is essential in data preparation. In picture (b), only the head and the withers are visible, but the model still also provides coordinates for the sacrum (likelihood 4.17%), the left (likelihood 1.58%) as well as the right (likelihood 10.79%) back stifles (see Figure 4.8, frame number 6979), which are interpolated during the data preparation. For patient number six (image (a)), only the right back stifle and the sacrum are visible. Unfortunately, the model falsely predicted the head with a likelihood of 70.36% and the sacrum, which should have been predicted in this frame, only with a likelihood of 32.25%, as can be seen in frame 1896 in Figure 4.8. Since these false predictions have likelihood values greater than the specified threshold, they are not included in the interpolation step during the data preparation. To get rid of these predictions anyway, a pre-filter followed by a Hampel filter were applied to the data after the interpolation step.

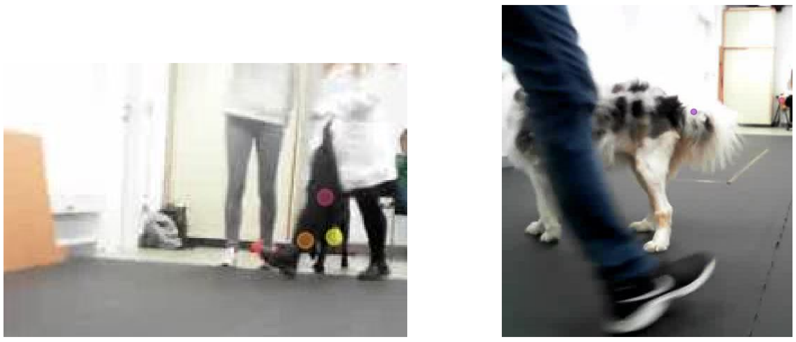

(a) Patient six – frame 00328 (b) Patient seven – frame 10222

Figure 4.11: Labeled frames with bad predictions: In image (a), the doctor obscures the dog while walking to the other side at the beginning of the video, leading to incorrect predictions by the model. Similarly, in the right image of patient number seven, the leg of the owner leading the dog during the analysis partially covers the dog, making it difficult for the model to identify the body parts.

As besides the dogs, the physicians who led the experiments and the owners who guided the patients are also visible, the videos include passages in which the dogs are partially covered by a human. Consequently, the model has problems identifying the body parts in these sections of the videos. This problem can be observed in the two images (a) and (b) of Figure 4.11, where in both cases, the leg of a human obscures parts of the dog. In the case of patient six, the model predicted the sacrum and left back stifle along the human leg with quite high likelihood values of 75.71% and 77.41% (see line 328 in Figure 4.8). In patient number seven, the human's leg covers the dog in such a way that the model predicted the withers (likelihood 94.27%) where the sacrum is located, and the sacrum was predicted at a spot next to the seated human in the background with a likelihood of 65.48% (frame 1022 in Figure 4.8). As already mentioned above, these outliers are filtered with the pre- and Hampel filters during data preparation.

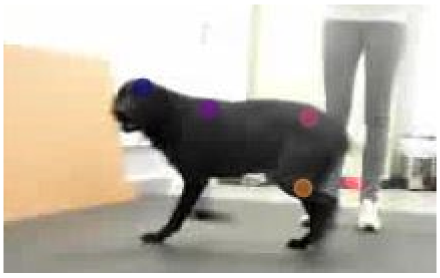


Figure 4.12: Labeled frame showing an ideal position: The dog is in a 90° angle to the camera position, so its side and therefore the head, withers, sacrum, and the respective back stifle are clearly visible.

With the algorithm's achieved cross-entropy loss of 0.00636, the interpolation of the body parts with likelihoods below 60%, and the extraction of wrongly predicted body parts (outliers) with likelihoods greater than 60% via pre- and Hampel filters, it was possible to extract accurate features from the signals. However, the performance of the algorithm could have been even better if, on the one hand, only the patient would have been seen in the videos and, on the other hand, the camera position had been chosen differently. An example of such an "ideal" position is shown in Figure 4.12, which was obtained while patient number six was turning around in the video. With the camera positioned at

90° towards the dog, the head, withers, sacrum, and respective back stifle would have been clearly visible and could have been labeled much more easily and accurately, resulting in a more precise training data set and a better trained model.

4.2. XGBoost classification results

This section presents and discusses the results of the parameter study, which uses different feature combinations as input to the classifier, as well as the performance of the XGBoost on new (unseen) data. For both cases, the model with the following hyperparameters was used:

Train-Test split	N_ estimators	Max_ depth	Learning_ rate	Gamma	Min_ child_ weight	Sub-sample	Col-_ sample bytree	Random_ state
0.2	263	2	0.24	0	0.65	0.12	0.52	5

Table 4.2: Hyperparameter settings for the XGBoost classifier.

Parameter study

The results of the parameter study are listed in Table 4.3, whereby the features used in each case, which were already listed in Chapter 3.3.2, are mentioned again for the sake of completeness. In order to ensure good readability, the features have been abbreviated as follows: H - FundHead, W - FundWithers, S - FundSacrum, M - Movement, Ma - Mass, We - Weight, F - Forces.

Feature combination		Performance		Accuracy
Number	Features	Train	Test	(%)
1	M, Ma, We, H	0.97	0.94	95.24
2	M, Ma, We, W	0.97	0.94	95.24
3	M, Ma, We, S	1.0	1.0	100.00
4	M, Ma, We, H, W	0.97	0.94	95.24
5	M, Ma, We, H, S	0.94	0.83	85.71
6	M, Ma, We, W, S	0.97	0.89	90.48
7	M, Ma, We, H, W, S	0.97	0.89	90.48
8	M, Ma, We, F	1.0	1.0	100.00
9	M, Ma, We, H, F	1.0	1.0	100.00
10	M, Ma, We, W, F	1.0	1.0	100.00
11	M, Ma, We, S, F	0.99	0.96	95.24
12	M, Ma, We, H, W, F	1.0	0.85	85.71
13	M, Ma, We, H, S, F	1.0	1.0	100.00
14	M, Ma, We W, S, F	1.0	1.0	100.00
15	all	0.99	0.94	95.24

Table 4.3: Results for the different feature combinations showing the calculated train and test performance (rounded to two comma places) as well as the accuracy for each combination.

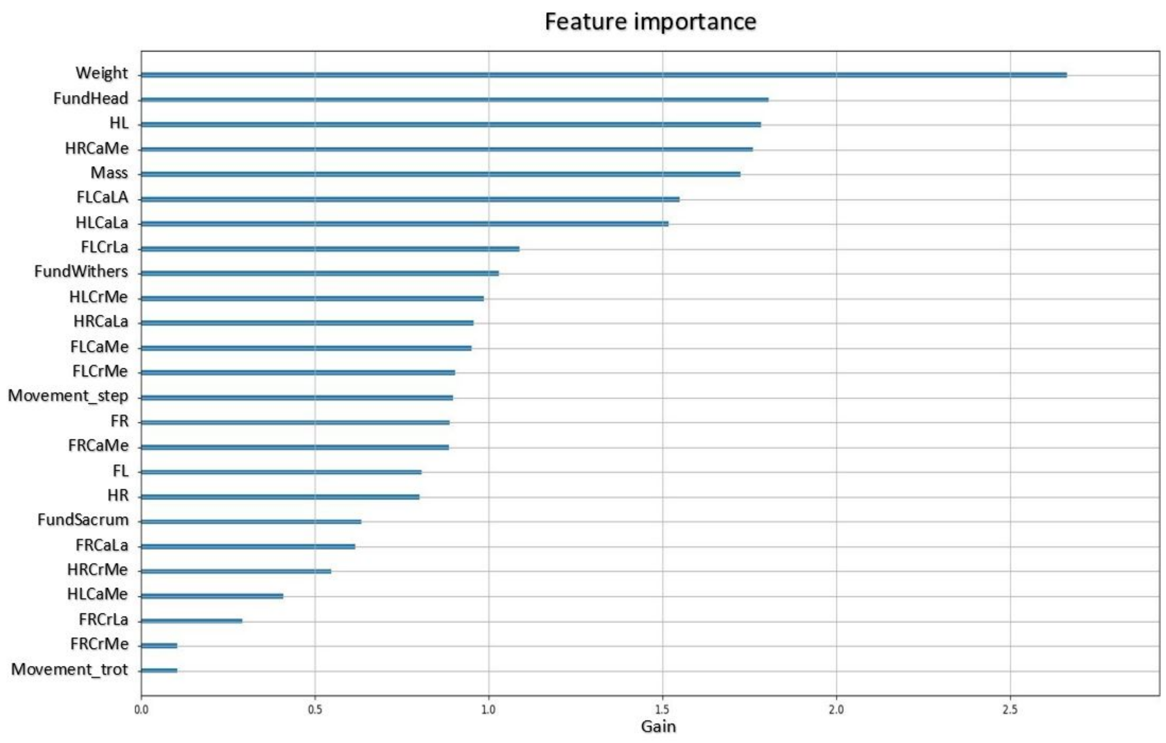


Figure 4.13: Feature importance

The diagram in Figure 4.13 illustrates the importance of each feature, which is provided by the "xgboost.plot_importance" function. For the importance_type parameter, the 'gain' was selected that averages the gain across all splits the feature is used in. The higher this value, the more important the feature is for the predictions.

Note: As the features mass, weight, and movement are used in each combination, they are not explicitly mentioned in the following discussion.

As can be seen in Table 4.3, the combinations one, two, and four achieved the same results, with a training performance of 0.97, a performance on the test set of 0.94, and an accuracy of 95.24%. Basically, these are very good results, and since the training performance is slightly better than the performance on the test data, no overfitting occurred. However, these sets using only the head and withers are not suitable for this application, as some types of lameness are evident only in the sacrum. An accuracy of 100% as well as test and training performances of 1.0 could indicate that the data set provides too little variance for the combinations three, eight, nine, ten, 13, and 14. For example, using only the sacrum (combination three) results in a problem, whereas using only the head (combination one) or withers (combination two) does not, meaning that perhaps not all patients diagnosed as diseased exhibit problems at the sacrum and therefore the data for this feature is not diverse enough.

However, when the sacrum is combined with other features such as the head, withers, and forces, this problem does not occur because (almost) all of these features are of greater importance to the classifier, as is shown in Figure 4.13. This is evident from the results in Table 4.3, where combination five (head and sacrum) has an accuracy of 85.71%, combinations number six (withers and sacrum) and seven (head, withers, and sacrum) have an accuracy of 90.48%, and combination eleven (forces and sacrum) achieved 95.24%. Numbers five and six would also not be well suited for this application, as, once again, they do not include all three body parts. Additionally, they did not reach a good accuracy and have a greater discrepancy between train and test performance, which could indicate possible overfitting. This problem also occurs in combination seven, where the training performance is 0.97 but the test is only 0.89, meaning that the model works well on the training data but not so well on the unseen data. Thus, in order to train a classifier efficiently using only the movements of the head, withers, and sacrum as input features (apart from weight, mass, and movement), a much larger data set would be required. Very good results were achieved with the combination of the features sacrum and forces, with a training performance of 0.99, a test performance of 0.96, and, as already mentioned, an accuracy of 95.24% (see Table 4.3, combination eleven), whereby, once again, only one of the three movement features was used here.

Quite interesting results can be seen in Table 4.3 for combination twelve (head, withers, and forces), which only achieved an accuracy of 85.71% with a training and test performance of 1.0 and 0.85, respectively. This is a perfect example of overfitting, as the model performs perfectly on the training data but poorly on the test data. With train and test performances of 0.99 and 0.94 and an accuracy of 95.24%, the model achieved pretty good results while using all extracted features as input (see Table 4.3, combination 15). The difference between the training and test performance is very small, which is why no overfitting has occurred yet. Nevertheless, a train performance of 0.99 indicates that further training with the same data set could cause the model to overfit, i.e., in order to improve the accuracy, the data set has to be extended first.

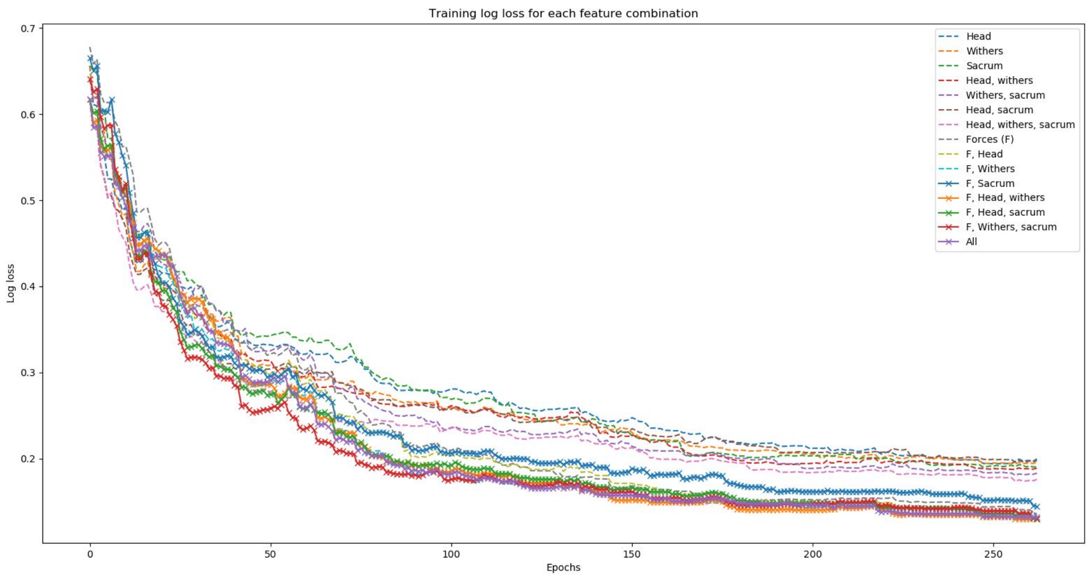


Figure 4.14: Log losses of train data sets

Figure 4.14 illustrates the log losses and provides a good overview of the performance of the model during the training for each feature combination. The x-axis represents the number of estimates, which are referred to as epochs. It can be seen that the feature selection has no significant effect on the log loss for the first 50 to 100 epochs, as it decreases at the same rate for all combinations. However, as epochs increase, the importance of feature selection becomes significant. At the end of the training, all combinations that included the forces achieved better log losses (all below 0.2) than those that did not, meaning that combinations that included only the head, sacrum, and withers (again, in addition to the mass, weight, and movement that are always part of the combinations) would require more epochs to achieve similar results.

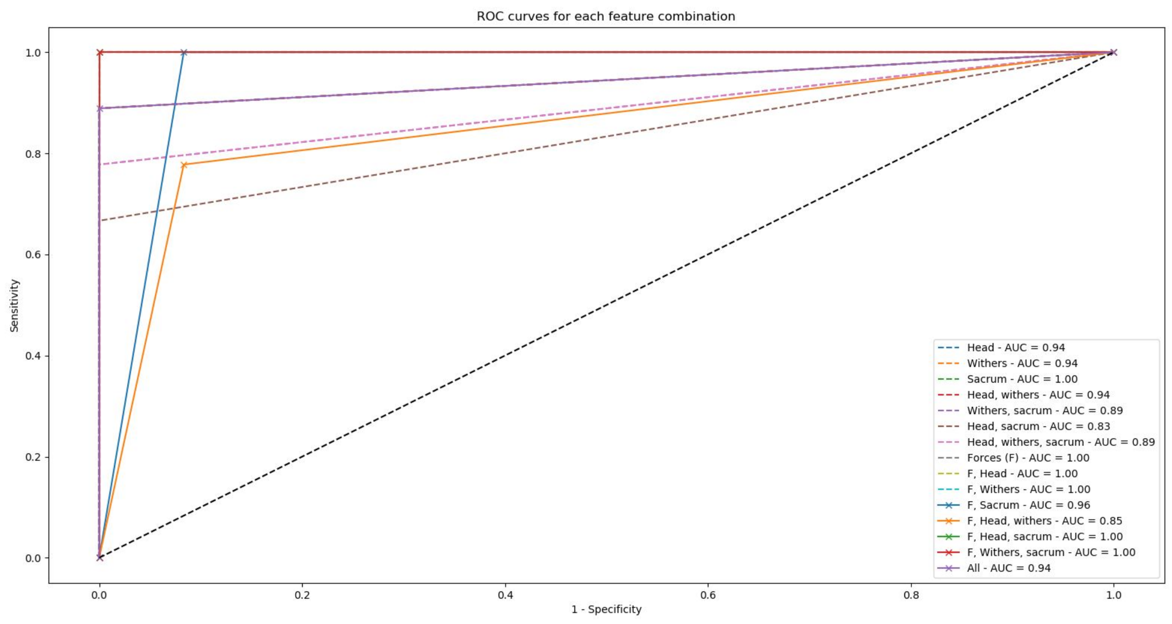


Figure 4.15: ROC curves and AUC values for each combination

In Figure 4.15, the ROC curves including the AUC values for each of the feature combinations are presented. The curve plots the true positive rate (also called sensitivity) against the false positive rate (1 - specificity) at different thresholds. The black dashed line denotes a kind of "baseline" where a random classifier would be expected to yield points. The closer the curve of the classifier is to that line, the less accurate the classifier. For a good comparison of the different feature combinations, the AUC was used, which summarizes the performance of each combination into one single measure and works well as a broad indicator of prediction accuracy. A value of one indicates a classifier which is able to distinguish perfectly between all the positive and negative class points, while a zero would mean that the classifier predicts all negatives as positives and vice versa.

As visible in Figure 4.15, none of the curves are close to the baseline, so the classifier is basically performing quite well for all feature combinations. The lowest AUC values ranged from 0.8 to 0.9 and were calculated for combinations five, six, seven, and twelve, which matches the accuracy results, where these four sets also performed the worst (see Table 4.3). For combinations where the accuracy value is 95.24% (one, two, four, eleven, and 15), the AUC score is either 0.94 or 0.96, which indicates a nearly perfect distinction between the two classes healthy or diseased. For those feature combinations where an accuracy of 100% is achieved because the variance of the corresponding feature data is not sufficient, an AUC value of 1.0 is calculated as expected.

XGBoost performance on unseen data

The following table summarizes the number of incorrect and correct predictions as well as the accuracies achieved with XGBoost during and after the training process on unseen data.

Data used	Wrong predictions	Correct predictions	Accuracy (%)
Test data set	1	20	95.24
Patients 1 and 12	10	14	58.33

Table 4.4: XGBoost predictions and accuracy on unseen data: The first line presents the results for the test data set used to evaluate the model during the training process. The performance of the trained XGBoost on the new data from patients one and twelve are outlined in row two.

As already mentioned, during the training process, the data was split into a test and a training data set, with the entire data set first being shuffled and then randomly sliced with respect to the given split value of 80:20. This test data set is not part of the training and is used to evaluate the classifier during the training process on unseen data. The results are outlined in the first row of Table 4.4, where XGBoost predicts 20 out of 21 diagnoses correctly, thus achieving an accuracy of 95.24%[1].

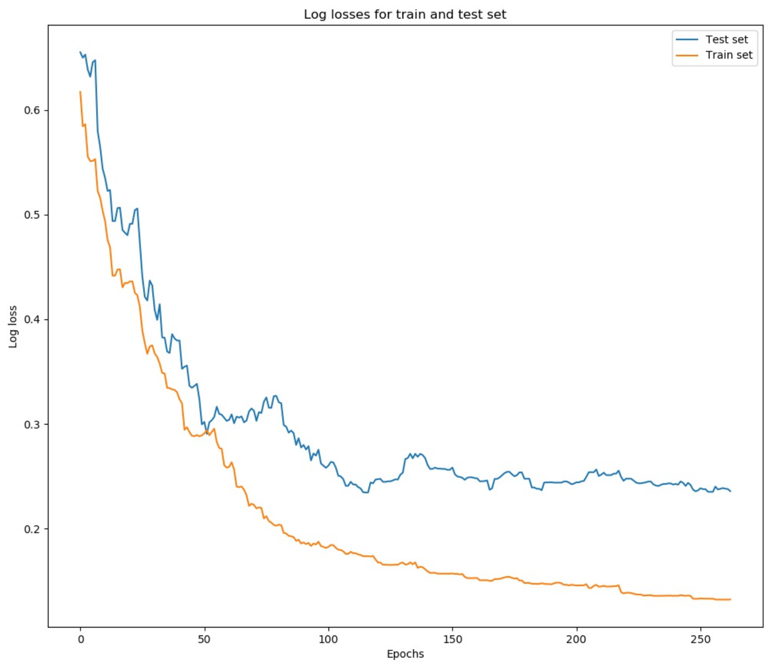


Figure 4.16: XGBoost log losses for train and test set

The calculated log losses for the train and the test set are plotted in Figure 4.16. In the training, the model reached a log loss value of less than 0.2 after 263 epochs (as already mentioned in the parameter study section), which means that the model makes very good predictions. For the test log loss, the value is only slightly higher, indicating that the model performs well even with unseen data, as reflected by only one wrong prediction out of 21 (see Table 4.4, row number one).

The final trained model was evaluated using data from patients one and twelve, who were entirely unknown to the classifier. The model incorrectly predicted ten out of 24 diagnoses and achieved an accuracy of only 58.33%. The fact that the two accuracy outcomes varied so much suggests that the final model does not generalize to new patients. To put it another way, the model does well on previously unknown data that was extracted from the training data set (test data set), but it does poorly on entirely new data (as shown with patients one and twelve). Generalization usually requires a huge amount of curated data [41].

[1] Note: Since the training was done with the same model, data set, and data split, the results are the same as those of the parameter study.

4.3. The gait analysis environment

As already noted in the previous sections, the lab situation during the gait analyses was not ideal. The camera's position, for example, was placed at an angle at one of the mat's end, thereby the dogs ran diagonally towards and back away from the camera. As a result, the patient's size vary throughout the video, i.e., the closer the dog gets to the camera, the larger it appears. Regardless of the position in the room or whether the dog is lame or not, each patient has distinct motion patterns that the head, withers, and sacrum follow when walking or running (with the exception of special occurrences such as sniffing). To recall: the DLC algorithm outputs the x and y coordinates of each body part per frame, which are used to calculate the vectors to represent the motion of the head, withers, and sacrum. Each movement leads to a difference between the coordinates of the previous frame and those of the current frame, resulting in vectors of different lengths and thus in signal jags. When the dog is near the camera, a movement leads to a large discrepancy in this coordinates, while a comparable movement results in very small differences when the dog is far away. Thus, the signal jags are of different sizes for similar movements, which in turn lead to an incorrect representation of the motion signal. Without correction, the XGBoost classifier, whose input includes features extracted from these signals, would have been trained in a non-generalized way; specifically, the model would only able to correctly classify new patients when the exact same camera position is used in the gait analyses.

By scaling the x and y coordinates as described in Section 3.3.1, the data used as input for the XGBoost model is independent of the camera position used during the gait analyses, in other words, the classifier does not care about the position of the camera. Nevertheless, with respect to the purpose of this application, the camera should not be placed at the pressure plate's ends, because in this case, the dog's side is not captured. For example, if the dog is walking straight towards the camera, only its head is visible and the withers as well as the sacrum are obscured, so that a proper observation of these two movements is not possible. If the camera is positioned at a 90° angle to the pressure plate, the scaling correction is completely obsolete, as the dog now appears in the same size throughout the video.

During the gait analysis, the dogs run several times towards the camera, turn around, run back away from the camera, and turn around again. Another issue brought on by the camera's position is the fact that one of these "turns" happens right next to the camera. Hence, there are sections of the video where the patients are not fully visible, which means that information about the body part that is not visible is lost. Since the turn occurs relatively quickly, always affecting only a few consecutive frames, interpolation can be used to fill in these information gaps. However, by increasing the distance between the camera and the pressure plate, it is possible to ensure that the dog's second turn also takes place entirely within the camera's field of view, preventing such a loss of data.

Besides the selected camera position, the fact that the patients were being guided by their owners was not optimal either. As a result, in some sections of the video, the dog was partially covered by a human body part. Basically, DLC offers two options for handling such hidden body parts: either only the visible body parts are labeled, in which case the algorithm learns to identify only the visible ones, or the positions of such hidden body parts are "guessed" during the labeling process, in which case the algorithm learns to detect covered parts as well. However, it is not advised to "guess" the positions of body parts because it is hard to mark such hidden body parts consistently and properly, which makes the algorithm's training more challenging and thus affects accuracy. Therefore, for this gait analysis application, occluded body parts were not labeled, which means that the trained algorithm does not identify them either. Interpolation is employed to restore the lost information, just as it was for the issue described in the previous section. For a more efficient training and hence an even better accuracy of the algorithm, videos where only the patient appears are advantageous.

5. Conclusion

The three main milestones in the development of the “AIDog Project” were the training and validating of the DLC algorithm, the feature extraction (including signal processing and data wrangling), and the training and validating of the XGBoost classifier. For the DLC part, the first step was to convert the MKV videos to AVI format, because the algorithm does not support this type. Since the provided data set is quite small, transfer learning was utilized in order to achieve better results. Thus, the weights of the algorithm were initialized with the weights of the open-source pre-trained “full dog” model provided by the model zoo. After seven training iterations with a total of 66000 learning iterations, the training process of the DLC was finished. Test data were used to evaluate the model after each training iteration, and based on the results, the training data set was extended and/or the learning rate was adjusted. The training data set included 480 frames initially before growing to 640 frames after the second training iteration and to 800 frames after the fourth. An outlier extraction was carried out following the sixth training iteration, where DLC extracted frames from the selected videos where the body part coordinates exhibited a significant difference from those in the previous frame. The labels of these frames were manually adjusted and the corrected frames have been added to the training data set, resulting in 1160 frames for the last training iteration. The final model was validated with the videos of patients number six and seven, and achieved an excellent cross-entropy loss of 0.00636 as well as a train error of 4.76 px and a test error of 5.45 px. It should also be mentioned that the DLC algorithm can be trained for any species, with model zoo already providing a variety of pre-trained open-source models.

The output files of the DLC, which contain the predicted x and y coordinates and their likelihood per body part and frame, were used to extract the fundamental frequencies of the patients’ head, withers, and sacrum movements. Here, to eliminate false predictions, each x and y coordinate, whose likelihood value was below the specified threshold of 60%, was interpolated. Afterwards, outliers were extracted applying a pre filter (sigma = 3) followed by a Hampel filter (sigma = 3, window size = 25). In the next step, scaling the x and y coordinates fixes the issue that the patient appears in various sizes throughout the video due to the camera position. Additionally, the vectors were calculated with the middle of the frame as the origin for the x-axis to ensure equal distances to the left and right edges[1]. Subsequently, the signals were smoothed using a one-dimensional Gaussian filter (sigma = 1), and a third-order high-pass filter with a cutoff frequency of 1 Hz ensures that irrelevant frequencies are removed. Finally, the signals were transformed using the fast Fourier transform and the fundamental frequencies for the head, withers and sacrum were extracted. From the provided Excel files that contain the data measured by the pressure plate, 22 features were extracted. These included the averaged forces for each paw, the averaged forces for each of the four areas per paw (CrLa, CrMe, CaLa, and CaMe), the mass and the dog’s weight. An additional feature was added to distinguish between the step/walk and the trot analyses. Since the XGBoost does not allow non-numerical columns, one-hot encoding was used to transform this feature into a numerical one. For training purposes, the target column with the physician’s diagnoses has been added, where an one is set if the dog is lame and a zero if the dog is healthy.

Before the XGBoost classifier training started, the randomized search function (with cross-validation) was used to identify the most suitable hyperparameters with respect to the input data. Here, the focus was set on the seven most frequently used ones: n_estimators, learning_rate, max_depth, gamma, min_child_weight, subsample, and colsample_bytree. The importance of feature selection as

[1] Note: DLC has its origin in the upper left corner.

well as the diversity in the data within each feature is demonstrated in the parameter study, which was conducted as part of the training. Results for six of the 15 feature combinations indicate a lack of diversity in the data, as both test and training performance scored a 1.0. To recall, a value of 1.0 means that the model differentiates perfectly between the classes, i.e., each case is classified correctly. The fact that both test and training data exhibit this excellent performance reflects an overly similarity in the data sets. Four other combinations caused the classifier to overfit. Only five of the 15 feature combinations yielded in well-performing models. These findings demonstrate how a classifier's performance is strongly influenced by the features that are applied to it. The final XGBoost model achieved 95.24% accuracy for previously unseen data, i.e., the test data set extracted from the training data set (patients two through eleven) that was not part of the training. However, applying this model to the completely new data of patients one and twelve yields an accuracy of only 58.33%, indicating that the trained model does not generalize well. Since, as already mentioned in previous sections, generalization usually requires a lot of curated data, it is necessary to extend the training data set to obtain a well-generalized XGBoost model for this application.

6. Literature

[1] E. Schnabl-Feichter, A. Tichy, and B. Bockstahler, "Coefficients of variation of ground reaction force measurement in cats," *PLOS ONE*, vol. 12, no. 3, D. Thamm, Ed., e0171946, Mar. 2017, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0171946.

[2] K.-D. Budras, P. H. McCarthy, W. Fricke, and R. Richter, *Anatomy of the Dog*, fifth, rev, K.-D. Budras, Ed. Schlütersche Verlagsgesellschaft mbH & Co. KG, Hans-Böckler-Allee 7, 30173 Hannover, 2007, ISBN: 978-3-89993-018-4.

[3] D. Leach, G. Sumner-Smith, and A. I. Dagg, "Diagnosis of lameness in dogs: a preliminary study.," *The Canadian Veterinary Journal*, vol. 18, no. 3, pp. 58–63, Mar. 1977, ISSN: 00085286.

[4] D. M. Nunamaker and P. D. Blauner, "Normal and abnormal gait," in *Textbook of small animal orthopaedics*, vol. 1083, International Veterinary Information Service New York, 1985, p. 1095, ISBN: 03-975-20980.

[5] B. J. Carr and D. L. Dycus, "Canine gait analysis," *Recovery & Rehab*, vol. 6, pp. 93–100, 2016.

[6] J. Blaszczyk and C. Dobrzecka, "Speed control in quadrupedal locomotion: principles of limb coordination in the dog," *Acta Neurobiol Exp (Wars)*, vol. 49, pp. 105–124, 1989.

[7] H. Scott and P. Witte, "Investigation of lameness in dogs," *In Practice*, vol. 33, no. 1, pp. 20–27, Jan. 2011, ISSN: 0263-841X. DOI: 10.1136/inp.c7447.

[8] P. Witte and H. Scott, "Investigation of lameness in dogs," *In Practice*, vol. 33, no. 2, pp. 58–66, Feb. 2011, ISSN: 0263-841X. DOI: 10.1136/inp.d453.

[9] A. Mathis, P. Mamidanna, K. M. Cury, *et al.*, "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning," Tech. Rep. 9, 2018, pp. 1281–1289. DOI: 10.1038/s41593-018-0209-y.

[10] Ł. Kidziński, B. Yang, J. L. Hicks, A. Rajagopal, S. L. Delp, and M. H. Schwartz, "Deep neural networks enable quantitative movement analysis using single-camera videos," *Nature Communications*, vol. 11, no. 1, 2020, ISSN: 20411723. DOI: 10.1038/s41467-020-17807-z.

[11] M. W. Mathis and A. Mathis, *Deep learning tools for the measurement of animal behavior in neuroscience*, Feb. 2020. DOI: 10.1016/j.conb.2019.10.008.

[12] Christoph Leitner and Stefan Kaltenböck and Christian Baumgartner and Markus Tilp, *Validation of an AI assisted simple method to study muscle-tendon dynamics during running*. International Society of Biomechanics, Jul. 2021.

[13] A. Mathis, S. Schneider, J. Lauer, and M. W. Mathis, *A Primer on Motion Capture with Deep Learning: Principles, Pitfalls, and Perspectives*, Oct. 2020. DOI: 10.1016/j.neuron.2020.09.017.

[14] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, Jul. 2020, ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2020.01.085.

[15] K. Yun, A. Huyen, and T. Lu, "Deep Neural Networks for Pattern Recognition," Sep. 2018.

[16] R. Bhalley, "Machine Learning Basics," in *Deep Learning with Swift for TensorFlow*, Berkeley, CA: Apress, 2021, pp. 1–35. DOI: 10.1007/978-1-4842-6330-3{_}1.

[17] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, Aug. 2018, ISSN: 18694101. DOI: 10.1007/S13244-018-0639-9/FIGURES/15.

[18] S. Ravichandiran, *Hands-on deep learning algorithms with Python*. Packt Publishing Ltd., Jul. 2019, ISBN: 978-1-78934-415-8.

[19] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, vol. 2018-Janua, IEEE, Aug. 2018, pp. 1–6, ISBN: 9781538619490. DOI: 10.1109/ICEngTechnol.2017.8308186.

[20] S. Sharma, S. Sharma, and A. Athaiya, "ACTIVATION FUNCTIONS IN NEURAL NETWORKS," *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, pp. 310–316, 2020.

[21] R. Bhalley, *Deep Learning with Swift for TensorFlow*. Apress, 2021. DOI: 10.1007/978-1-4842-6330-3.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Tech. Rep., 2016, pp. 770–778.

[23] Koki Saitoh, *Deep Learning from the Basics*. Packt Publishing Ltd., Mar. 2021, ISBN: 978-1-80020-613-7.

[24] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022 022, Feb. 2019, ISSN: 1742-6588. DOI: 10.1088/1742-6596/1168/2/022022.

[25] D. D. Jensen and P. R. Cohen, "Multiple Comparisons in Induction Algorithms," *Machine Learning*, vol. 38, no. 3, pp. 309–338, 2000, ISSN: 08856125. DOI: 10.1023/A:1007631014630.

[26] A. Mathis, T. Biasi, S. Schneider, *et al.*, "Pretraining boosts out-of-domain robustness for pose estimation," *Proceedings - 2021 IEEE Winter Conference on Applications of Computer Vision, WACV 2021*, pp. 1858–1867, 2021. DOI: 10.1109/WACV48630.2021.00190.

[27] E. Raff, *Inside Deep Learning*. Manning Publications Co., 2022, ISBN: 9781617298639.

[28] P. Singh, *Fundamentals and Methods of Machine and Deep Learning*. John Wiley & Sons, Inc. and Scrivener Publishing LLC, 2022, ISBN: 978-1-119-82125-0.

[29] C. Wade and K. Glynn, *Hands-On Gradient Boosting with XGBoost and scikit-learn*. Packt Publishing Ltd., Livery Place, 35 Livery Street, Brimingham B3 2PB, UK., 2020, ISBN: 978-1-83921-835-4.

[30] V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman, "Decision Trees: An Overview and Their Use in Medicine," *Journal of Medical Systems 2002 26:5*, vol. 26, no. 5, pp. 445–463, Oct. 2002, ISSN: 1573-689X. DOI: 10.1023/A:1016409317640.

[31] S. Kumar, T. Daniya, M. Geetha, and K. S. Kumar, "Classification and regression trees with gini index," *Advances in Mathematics: Scientific Journal*, vol. 9, no. 10, pp. 1857–8438, 2020. DOI: 10.37418/amsj.9.10.53.

[32] Y. Liu, Y. Wang, and J. Zhang, "New Machine Learning Algorithm: Random Forest," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7473 LNCS, pp. 246–252, 2012, ISSN: 03029743. DOI: 10.1007/978-3-642-34062-8{_}32.

[33] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. DOI: 10.1145/2939672.

[34] T. Nath, A. Mathis, A. C. Chen, A. Patel, M. Bethge, and M. W. Mathis, "Using DeepLabCut for 3D markerless pose estimation across species and behaviors," *Nature Protocols*, vol. 14, no. 7, pp. 2152–2176, 2019, ISSN: 17502799. DOI: 10.1038/s41596-019-0176-0.

[35] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele, "Deepercut: A deeper, stronger, and faster multi-person pose estimation model," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9910 LNCS, Springer Verlag, 2016, pp. 34–50, ISBN: 9783319464657. DOI: 10.1007/978-3-319-46466-4{_}3.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386.

[37] A. Zheng and A. Casari, *Feature Engineering for Machine Learning*. O'Reilly, 2018, ISBN: 978-1-491-95324-2.

[38] R. Lehmann, "3σ-Rule for Outlier Detection from the Viewpoint of Geodetic Adjustment," *Journal of Surveying Engineering*, vol. 139, no. 4, pp. 157–165, Nov. 2013, ISSN: 0733-9453. DOI: 10.1061/(ASCE)SU.1943-5428.0000112.

[39] A. B. Batista Júnior and P. S. M. Pires, "An Approach to Outlier Detection and Smoothing Applied to a Trajectography Radar Data," *Journal of Aerospace Technology and Management*, vol. 6, no. 3, pp. 237–248, Sep. 2014, ISSN: 2175-9146. DOI: 10.5028/jatm.v6i3.325.

[40] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A Comprehensive Survey of Loss Functions in Machine Learning," *Annals of Data Science*, vol. 9, no. 2, pp. 187–212, Apr. 2022, ISSN: 2198-5804. DOI: 10.1007/s40745-020-00253-5.

[41] C. Leitner, R. Jarolim, B. Englmair, *et al.*, "A Human-Centered Machine-Learning Approach for Muscle-Tendon Junction Tracking in Ultrasound Images," *IEEE Transactions on Biomedical Engineering*, vol. 69, no. 6, pp. 1920–1930, Jun. 2022, ISSN: 0018-9294. DOI: 10.1109/TBME.2021.3130548.

Appendices

A. AIDog Project

This section provides a brief overview of the Python scripts developed in PyCharm (JetBrains s.r.o., Prague, Czech Republic) according to the PEP-8 coding standard. All required site packages are provided in the Anaconda environment "AIDog-env.yaml"[1].

train_DLC

This sub-project contains all the scripts used to train the DLC algorithm. When working on Windows, make sure to open the project with administrator rights, otherwise DLC will have problems adding videos to the project.
Note: The following only offers a quick overview of the scripts, for the settings within the yaml files, please refer to the user manual provided by DLC: "Using DeepLabCut for 3D markerless pose estimation across species and behaviors" [34].

video_converter_re_encoder.py

Input: Video(s) used for the training or analyzing
Output: Re-encoded video(s) in AVI format

DLC does not allow the MKV video format, so first of all use this script to convert all videos to AVI. Since it is possible that the videos could get corrupted during this process, this script also re-encodes the videos. Note: If the video is already in the right format, it is recommended to use re-encoding anyway to avoid errors due to corrupted videos during the training or analyzing process.

train_utils.py

In this script the functions used by DLC for training are accessed. Before the main.py is executed, some parameters have to be set in the function "start_training". These include the number of iterations to be displayed and saved as well as the maximum number of iterations per training. In case of an adaptation of the body parts, the parameter "keepdeconvweights" must be set to false to delete the weights in the deconvolutional layer as described in Section 3.2.2, it is currently not possible to solve this in any other way.

main.py

Input: Videos for training
Output: Trained DLC

This is the main script for the training application and provides two possibilities: Either start a new project or open an already existing one. To start a **new project**, specify the name of the project and the experimenter, the path to the folder with the videos used for the training, and the path where the project should be created. Make sure that the parameters to open a new project are set to "None". If the videos should not be copied to this folder, set the copy_videos parameter to false and DLC will only create symbolic links [34]. If weights from the model zoo are used, they are automatically

[1] Do not update any packages, as DLC requires special versions for some of them, for example: *wxpython* $\leq$ 4.1.0.

downloaded and added to the project. After the parameters are all defined, including the parameters of the training function in train_utils.py as described above, the script can be executed and the full process flow starts.

With the provided project and experimenter names, DLC creates a new project folder with the config.yaml file and the subfolders. The body parts of interest, the quantity of frames to extract, and the train-test-split value are just a few examples of the parameters that are defined in this config.yaml file. Once these adjustments are complete, DLC extracts the frames and the experimenter can begin to label the extracted frames. With the labeled frames, DLC creates a data set and splits it into a train and a test data set. Before the training starts, the appropriate settings for the training are made in the pose_cfg.yaml file, which is located in the train subfolder. These include, for example, the learning rates and the initialization of the weights. The training progress is stored in the train subfolder, which can be found in the corresponding training iteration folder (dlc models). [34]

To **open a project**, simply provide the path to the project folder and the starting point in the script. The starting points can be "extract", "label", "dataset" or "train". Depending on the selected point, the process flow starts with the corresponding phase of the process flow. Each time an existing project is opened, the experimenter is asked whether new videos should be added. To do this, specify the folder path of the new videos before running the script and type "Yes" in the command window. The whole process flow will start automatically and the frames can be extracted from the videos. During this process, DLC will ask the experimenter for each of the videos individually, so it is also possible to extract frames for specific videos only [34]. This procedure is very handy if, for example, only frames from the new videos should be added and no additional frames from the already used videos.

network_evaluation.py

Input: DLC model to evaluate
Output: Evaluation results as CSV file

This script is used to evaluate the trained model. Unless otherwise defined, DLC uses the last snapshot saved during the last training iteration. This can be changed by adapting the two parameters "iterations" and "snapshotindex" in the config.yaml file of the project. For the iterations, simply set the number of the wanted training iteration, for the snapshot index it is possible to either evaluate all of them (set parameter to all), only the last one (set parameter to -1) or a specific one from the stored snapshots (set parameter to specific number[2]). For the results, DLC creates the subfolder "evaluation-results" where the calculated MAE values are stored as a CSV file and, if the plotting parameter is set to "True", the frames used for the evaluation can be found. [34]

analyze_videos.py

Input: Video(s) to analyze
Output: DLC predictions (x and y coordinates with likelihood as XLSX file); labeled video

For the analysis of the videos, specify the training iterations and the snapshot index to be used in the config.yaml file (as described for the network evaluation). The x and y coordinates including their likelihood values are stored in an XLSX file for each video. Optionally, it is also possible to create labeled videos.

[2] Note: DLC refers to a list index, not the snapshot number, i.e., if there are 5 stored snapshots, use snapshotindex 3 to evaluate the fourth stored snapshot.

network_refinement.py

Input: Analyzed video(s) with DLC predicted labels
Output: Re-labeled frames added to the training data set

This script takes the video(s) provided and extracts outliers based on the defined parameters. The number of frames to be extracted is specified in the config.yaml file.The experimenter can review the labels generated by the algorithm for each extracted frame and change or remove them as needed. Note: Be careful when deleting labels, this step cannot be undone. Save the relabeled frames and load the next file until each file is relabeled. Then, DLC merges the newly labeled images with the existing data set and creates a new training iteration (the iteration parameter in config.yaml is automatically incremented). [34]
After that, a new training can be started by executing the main.py script.

process_flow.py

In this script all functions used for the process flow are specified. The full process flow is defined as follows [34]:

- DLC creates new directory
- Experimenter adapts config.yaml file
- DLC extracts frames
- Experimenter labels extracted frames
- DLC creates data set
- Experimenter adapts pose_cfg.yaml file (in train folder of current training iteration)
- DLC starts training

In the case of a new project and the starting point "extract", the full process will start. For the other starting points "label", "dataset", and "train", the process will start with the corresponding stage of the flow. After some of the steps, terminal input is required from the experimenter; this ensures that the experimenter has enough time to adapt the yaml files and to check the labeled images.

ML_classifier

DataWrangling_utils.py and SignalProcessing_utils.py

These two scripts contain all the functions needed for the data wrangling process, with DataWrangling_utils.py covering the functions required for the feature extraction from the PA files. Within this script, SignalProcessing_utils.py is accessed where the processing of the DLC files is performed. The features extracted during this signal process are returned and added to the data frame within the DataWrangling_utils.py.

train_XGBoost_dataprep.py

Input: Excel files containing pressure data and body part prediction data
Output: Data frame with features and averaged values

Since the Excel files from the PA tool and the DLC output share the same name, they must be provided in different folders. All files in the folders are extracted and stored in two different lists. The basic structure of the data preparation is to take one PA file, process its data, and assign the extracted features (averaged forces, mass and weight) to the data frame. The diagnosis and movement are identified by the name of the file and added as well. This is done for each pass (subset) of the PA data, with the extracted features of each subset stored in separate rows of the data frame. Afterwards, the DLC file with the same name, i.e., the same patient and the same gait analysis, is selected, the fundamentals extracted and assigned in the same row of the data frame. This completes the data processing for one patient, so the next patient's data preparation is started. For the average of the paws and their areas, a loop is used to process one paw at a time. At the beginning of the script, the numbers of the columns in the PA files are checked to identify if there is measured data of the areas for each measured paw, otherwise, the file is excluded from the process. With the parameter "pass_diff", the possibility is provided to create data frames without differing between the passes, i.e., if this parameter is set to "False", the entire measured data is averaged and each row within the final data frame represents one patient.

XGBoost_RandomizedSearch.py

Input: Data frame containing the processed data
Output: Most suitable parameters for the classifier

Search for the Most suitable hyperparameters using the "RandomizedSearchCV" function, provided by the sklearn library, by defining a set of values within a grid. For the search, the repeated stratified k-fold is used.

XGBoost_ParamStudy.py

Input: Data frame containing the processed data
Output: Accuracy, log loss, and ROC curve for each of the feature combinations

Within this script, the different feature combinations are defined and processed. For all these combinations, the same XGBoost model is used and the train and test performance as well as the accuracy values are printed. After all combinations went through the training of the classifier, the ROC curves, including their AUC values, and the log losses computed during the training are plotted.

train_XGBoost.py

Input: Data frame containing the processed data
Output: Accuracy, log loss, and ROC curve

Specify the path to the saved data frame containing the features to be used and set the hyperparameters for both the XGBoost classifier and the test-train-split before running the script. After the training is completed, the log loss and ROC curve diagrams as well as the accuracy score for the trained classifier are displayed and the trained classifier is stored as DAT file.

test_XGBoost.py

Input: Data frame containing new data
Output: Accuracy, predicted and true diagnoses

This script loads the trained XGBoost model and tests it using a data frame containing new data. The predicted and the true diagnoses as well as the achieved accuracy are printed in the terminal.

Final_Program.py

Input: PA Excel file and video of the patient
Output: Diagnosis

This script combines the analysis of the video and the classification parts and is used to classify "new" patients. For the input, it is important that the PA file names still contain the letters for differing between the trot "_T_" or the step/walk "_S_", as this is hard coded in the script. Put the pressure data file and the recorded video in the same folder and simply provide the path to it. First, the video will be analyzed using the trained DLC algorithm. The training iteration number as well as the used snapshot index will be displayed in the terminal, so it is possible to cross-check. In case another model should be used, simply change the snapshot index and the training iteration index in the config.yaml file in the project folder. After this, the data preparation of the pressure data and the predictions starts using the entire measured data, so there is no differentiation between the passes. The data frame is saved as an Excel file, so in case of any problems or doubts, it is possible to look at the processed data. Finally, the trained and saved XGBoost model is loaded and the classification of the patient is displayed in the terminal.

B. List of Figures

C. List of Tables