Lauri Kantola

Tracking a rat in an open field experiment with a deep learning-based model

Master's Thesis in Information Technology

October 8, 2021

University of Jyväskylä

Faculty of Information Technology

Author: Lauri Kantola

Contact information: `lauri@kanto.la`

Supervisor: Paavo Nieminen, Miriam Nokia

Title: Tracking a rat in an open field experiment with a deep learning-based model

Työn nimi: Rotan seuraaminen avoimen kentän kokeessa syväoppimiseen perustuvan mallin avulla

Project: Master's Thesis

Study line: Software and Telecommunication Technology

Page count: 45+6

Abstract: New artificial neural network methods have changed the way animals are tracked in neuroscience and psychology experiments. The purpose of this thesis is to test the state-of-the-art method of animal tracking DeepLabCut and to develop a usable model for tracking rats in an open field type experiment, and to use tracking information to extract research-related key figures via self-tailored analysis software. The model trained with the DeepLab-Cut and 825 labeled images was accurate and suitable to be used in a research experiment. With the help of tracked body parts, it was possible to extract meaningful key figures for further analysis in a research experiment.

Keywords: animal tracking, artificial intelligence, artificial neural networks, convolutional neural networks, DeepLabCut, deep learning, neuroscience, machine learning, machine vision, open field test, rat, rodent

Suomenkielinen tiivistelmä: Uudet keinotekoisiin hermoverkkoihin perustuvat menetelmät ovat muuttaneet sitä, miten eläimiä seurataan neurotieteen ja psykologian koeasetelmissa. Tämän tutkimuksen tarkoituksena on ollut testata neuroverkkoihin perustuvaa DeepLabCut-menetelmää rottien seurantaan avoimen kentän testin tyyppisessä koeasetelmassa ja tuottaa seurantadatasta tutkimusten kannalta merkityksellisiä avainlukuja itse kehitetyn analyysiohjelman avulla. DeepLabCutilla ja 825 kuvalla opetettu malli oli tarkka ja soveltuva tutkimuskäyttöön. Seurattujen eläimen kehopisteiden avulla pystyttiin tuottamaan tunnuslukuja tutkimusten analyysejä varten.

Avainsanat: avoimen kentän testi, DeepLabCut, eläimen seuraaminen, jyrsijä, keinotekoiset neuroverkot, konenäkö, koneoppiminen, konvoluutioneuroverkot, neurotiede, rotta, syväoppiminen, tekoäly

Preface

Completing this thesis was a long haul, but it was never forgotten and thus completed in the end. I am grateful to my supervisor Paavo Nieminen for his time, persistence, and understanding during this process. I also want to thank Miriam Nokia for supervising and providing training videos and interest in this thesis.

Special thanks to Mirjami and our dear dog Mona for supporting me during this phase.

In Laukaa October 8, 2021

Lauri Kantola

List of Figures

Figure 1. Different open field types 4
Figure 2. Context-object discrimination 6
Figure 3. Simple perceptron 8
Figure 4. Feedforward neural network 9
Figure 5. Single node structure in hidden layer and output layer 10
Figure 6. ReLU activation function 11
Figure 7. Convolutional neural network image classification 12
Figure 8. ResNet "bottleneck" structure 16
Figure 9. DeepLabCut ResNet-50 19
Figure 10. Annotated body parts 21
Figure 11. The model trained with the default parameters and ResNet-50 base 23
Figure 12. The model trained with the custom parameters and ResNet-50 base 24
Figure 13. The model trained with the custom parameters and ResNet-101 base 25
Figure 14. Rodent Open Field Tracker analysis visualized 28
Figure 15. Configuring objects in the configuration step 29
Figure 16. Editing object shape in the configuration editor 30

List of Tables

Table 1. 19-layer VGG architecture 15
Table 2. 50-layer ResNet architecture 16
Table 3. Validation dataset accuracy in pixels per body part. 25
Table 4. Validation dataset accuracy in pixels per video and body part. 26
Table 5. Details of the dataset videos. 42

Contents

1 INTRODUCTION ... 1

2 BACKGROUND ... 3
- 2.1 Tracking methods in behavioral neuroscience ... 3
- 2.2 Experimental designs ... 3
  - 2.2.1 Open field test ... 3
  - 2.2.2 Context-object discrimination ... 5
- 2.3 DeepLabCut ... 6
- 2.4 Deep learning ... 7
- 2.5 Network types and structures ... 8
  - 2.5.1 Feedforward neural networks ... 9
  - 2.5.2 Convolutional neural networks ... 12
  - 2.5.3 ImageNet ... 14
  - 2.5.4 VGG ... 14
  - 2.5.5 ResNet ... 15
- 2.6 From DeepCut to DeeperCut and DeepLabCut ... 17

3 NEURAL NETWORK MODEL TRAINING ... 20
- 3.1 Dataset ... 20
- 3.2 Training ... 21
- 3.3 Results ... 22

4 USING TRACKING IN A PRACTICAL APPLICATION ... 28

5 DISCUSSION ... 31

BIBLIOGRAPHY ... 35

APPENDICES ... 40
- A Dataset videos ... 40
- B ROFT per video configuration ... 43
- C ROFT project-wide configuration ... 44
- D ROFT extracted figures ... 45

1 Introduction

The ability to track the movement and behavior of an animal is fundamental in behavioral neuroscience, but also in many other fields of science that use animals in research experiments. For this reason, there have been numerous methods invented for tracking animal movement and behavior throughout the history of psychology and behavioral neuroscience. In the early days, the methods were laborious due to the absence of suitable technology like advanced artificial intelligence, powerful computers, and high-quality digital cameras. Later, while computers became more common, many computer vision methods have been developed for tracking animals. To this day, computer vision methods solely rely on well-defined mathematical algorithms. Though these algorithms work fine with simple explicit tasks and are rather efficient, the problem is the difficulty of developing the algorithms in addition to matching the accuracy of the human eye on complex visual tasks. Recent progress in artificial neural networks has shed new light on tracking animals in behavioral neuroscience research. Artificial neural networks enable researchers to train neural models to track the movement and the behavior of animals in reasonably complex environments with just a thousand pre-labeled training data. Recent research (Mathis et al. 2018) has shown that it is possible to get close to human accuracy tracking with a small amount of annotated data.

In this study, the state-of-the-art tool DeepLabCut (Mathis et al. 2018) is tested for animal tracking in an open field type setting with a heterogeneous collection of pre-recorded videos of open field experiments with a single Sprague Dawley rat. Due to the laborious nature of the previous methods, this method is a long-awaited improvement as an automated animal tracking tool to be used in behavioral neuroscience experiments in Miriam Nokia's research group at the Department of Psychology at the University of Jyväskylä. The primary goal of this project is to enable the use of DeepLabCut in real experiments and to develop it further. Along with the primary goal, another objective is to test the DeepLabCut in animal tracking with local pre-recorded heterogeneous open field type video material, and assess the suitability of the selected method for tracking animals in an open field setting. This thesis gives a practical example and tools to utilize neural network-based tracking in open field experiments via specialized software that extracts meaningful key figures from an open field setting. As a result, a pre-trained model that can be used for tracking purposes is provided for the research group.

Research questions are as follows:

- RQ1: Is DeepLabCut suitable for tracking animals in an open field type setting?
- RQ2: How accurate is DeepLabCut compared to humans in detecting user-defined landmark points in a similar but novel video material?
- RQ3: How can tracking results be utilized with the analysis software presented here?

Chapter 2 describes the relevant details in the context of open field experiments, neural networks, and DeepLabCut. Chapter 3 describes the process of creating the datasets, training, and evaluating the DeepLabCut models. Chapter 4 presents the custom software used to analyze tracking results further. In Chapter 5 the whole process is evaluated and the future of the neural network-based tracking assessed.

2 Background

2.1 Tracking methods in behavioral neuroscience

Numerous tracking methods have been developed to use in psychology and behavioral neuroscience. In behavioral neuroscience, tracking is often used as a tool for assessing animal learning or social behavior in various experiments. Tracking information of animal's location is also used to identify neurons in the brain firing in relation to animal's location in space. Discovery of place cells and grid cells led to Nobel prizes for John O'Keefe (place cells) and Edvard and May-Britt Moser (grid cells) (see Burgess 2014). Without advanced computer vision-based tracking these cells might have not been discovered.

Before the use of computer vision based methods, different electronic devices were used. For example, beams of infrared light, force measuring sensors or different types of touch sensors, ultrasound or microwave radars could have been used (see Noldus, Spink, and Tegelenbosch 2001; Spink et al. 2001). Computer vision methods later became the golden standard, but there was a lack of a highly accurate and generalizable ways to track animals or individual body parts reliably. The used methods relied on specialized environments and backgrounds, or painted markers on the animals and for example, used image color differences and background subtraction to detect and to track the animal (see Noldus, Spink, and Tegelenbosch 2001; Spink et al. 2001; Lopes et al. 2015).

2.2 Experimental designs

2.2.1 Open field test

Many experimental designs are based on an open field test (OFT). OFT was originally developed to assess rodent activity and emotionality (Denenberg 1969), but its key concept, an open field, was later adopted in numerous experimental designs. At present, OFT is an enclosure (usually square, rectangular, or circular), in which rodents (or animals of other species) move freely and animals' explorative behavior, activity, and other behaviors are measured in different ways (Gould, Dao, and Kovacsics 2009).

Gould, Dao, and Kovacsics (2009) listed common figures to be extracted from an open field test:

- Distance moved
- Time spent moving
- Standing on two feet (rearing)
- Freezing behavior
- Grooming behavior
- Abnormal/repetitive behavior
- Time spent in center
- Center crossings
- Defecation
- Urination

Field shape could be varied and markings, objects, sectors or areas could be added related to experimental design. By varying the open field it is possible to do a multitude of different experimental designs that for example are intended to measure animals' learning and memory. In Figure 1 is different open field designs as an example.

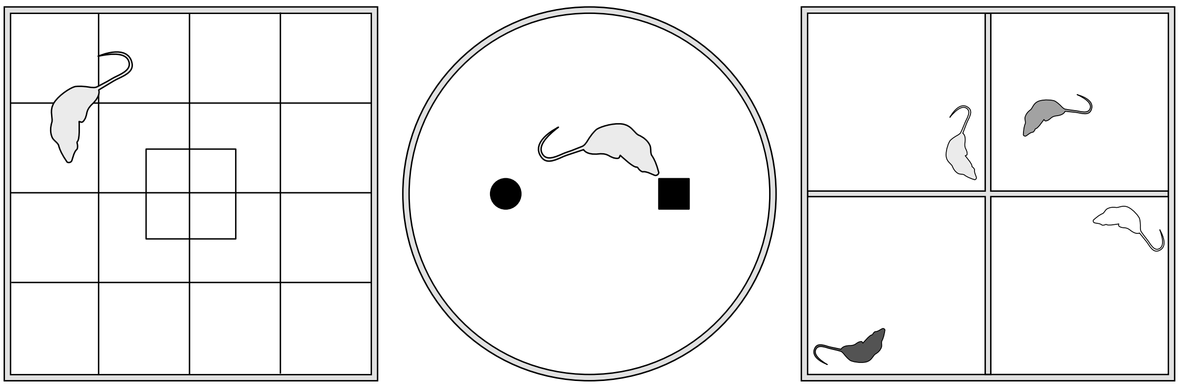


Figure 1. Different open field types. From left to right: a) square arena with grid sectors and center sector, b) circle arena with two different objects, c) multiple animals with individual compartments.

2.2.2 Context-object discrimination

Context-object discrimination (COD) task is an open field type test, which is used to study pattern separation and context related learning (see Lensu et al. 2019). Pattern separation is an ability to discriminate between different, but often similar stimuli (see Aimone et al. 2014; Kantola, Pirttimäki, and Nokia 2017). COD is also known as object-in-context (OiC) recognition (see Ramsaran, Westbrook, and Stanton 2016).

At Miriam Nokia's lab, COD has been used to study the role of the hippocampal dentate gyrus granule cells in pattern separation and formation of non-overlapping contextual memories. Lensu et al. (2019) (see Figure 2) used a square 74 cm x 74 cm open field arena with 30 cm high walls, and a circular 74 cm diameter arena, also with 30 cm wall. They created six different variations of the environment by decorating the walls to create visual cues, changing floor materials, and varying background visual cues and lighting. Typical small household items and other small artifacts were used as objects. Rats were put to three different contexts on different days, so that the first two contexts contained two of the same items, but were different between contexts. On the third day, the context was either one of the two, but one of the objects was switched with the other context object that was not selected for the third day context. The first and the second day contained two hours of brain stimulation related to the research question. It was assumed that the rats preferentially explored the so-called out-of-context object, the switched object. Experiments were recorded with regular web cameras, and exploration of the objects was later assessed by researchers. Researchers measured time spent less than 2 cm far from the object when the animal's nose was pointing in the object's direction, or the animal was physically interacting with the object. Lastly, relative time spent at the out-of-context object was calculated.

The idea for this Master's thesis originated from the need to track animal movement and measure time spent near objects in a COD task presented here. A DeepLabCut model was created to track rat body parts in an open field, and an analysis program was developed to extract figures of interest from the tracking data and videos.

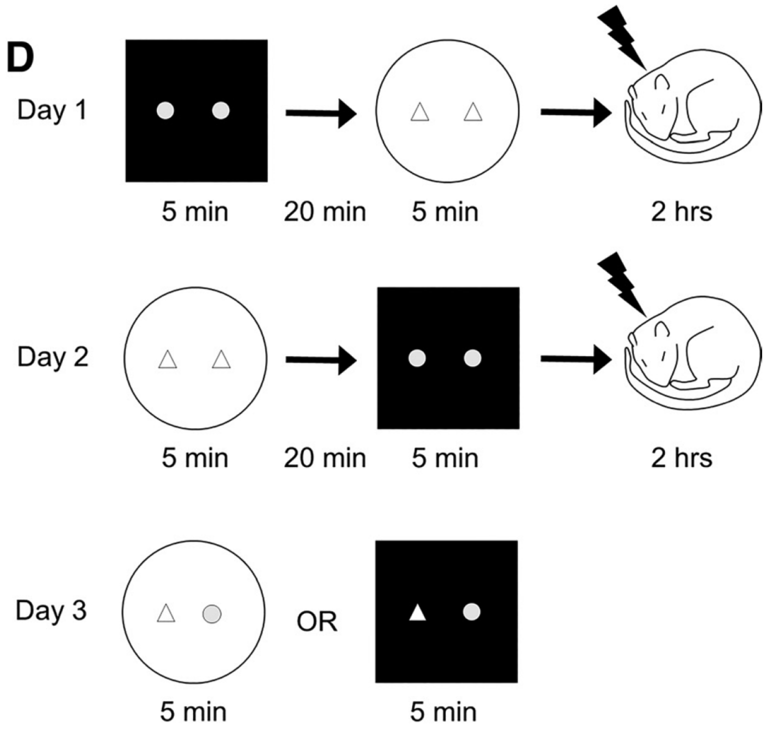


Figure 2. Context-object discrimination. Courtesy of Lensu et al. (2019).

2.3 DeepLabCut

DeepLabCut (Mathis et al. 2018) is a fairly recently developed Python toolbox for tracking virtually any animal species efficiently. With DeepLabCut, the user can track a set of user-defined landmark points, body parts, of an animal with a small amount of pre-labeled data.

DeepLabCut takes advantage of the state-of-the-art achievements in artificial neural network-based multi-person pose estimation methods (Pishchulin et al. 2016; Insafutdinov et al. 2016; Mathis et al. 2018). DeepLabCut provides a complete set of tools for creating and labeling datasets, training and evaluating models, running trained models against the experiment videos, and pruning results via various filtering methods to acquire meaningful information for later analysis. Also, DeepLabCut supports the plotting of the acquired data for further inspection or visualization. The toolbox is also usable via a graphical user interface (GUI) to some extent, but it is not required. A more detailed description of the body part detection used by DeepLabCut is given in Section 2.6.

2.4 Deep learning

Deep learning uses deep multi-layered structures of connected nodes with tunable parameters to learn complicated concepts by forming them via simpler concepts (see Goodfellow, Bengio, and Courville 2016, p. 1–2). Deep learning is nothing new, as one might think, but its foundation dates back to the 1940s, and the story began from a wish to understand the biological basis of learning and brain functions via modeling the neurons and neural circuits of the brain. Cybernetic researchers started to develop theories of biological learning of the brain in the 1940s, and the first models were implemented in the 1950s (Goodfellow, Bengio, and Courville 2016, p. 13–14).

The first models, like perceptrons, mimicked a single neuron that could learn input classifying weights from the given example inputs of each category (Rosenblatt 1958; Goodfellow, Bengio, and Courville 2016, p. 15). A single neuron contains typically multiple inputs called dendrites, a soma (cell body), and an axon that carries neural spikes (action potentials) to other cells' dendrites in the brain (Nolen-Hoeksema et al. 2009, p. 36–42; Charniak 2018, p. 3–5). As neurons, simple perceptrons (see Figure 3) have multiple inputs (dendrites) and a function (a soma) that outputs (an axon) a result of 1 or 0, depending on when multiplying each input weight with an input value and adding the bias is greater or less than 0 (Rosenblatt 1958; Charniak 2018, p. 3–5).

The era of cybernetics lasted till the 1960s, and the next significant wave occurred, when a movement called connectionism emerged in the 1980s (Goodfellow, Bengio, and Courville 2016, p. 14, 17). It lasted until 1995 during which connectionists were able to train networks with one or two hidden layers using backpropagation and stochastic gradient descent optimizer (Goodfellow, Bengio, and Courville 2016, p. 14–18). However, deep learning did not break to its peak until the current "deep learning" era started around 2006 (Goodfellow, Bengio, and Courville 2016, p. 14, 18-19). The researcher Geoffrey Hinton with his colleagues (2006) showed an efficient way to train a so-called deep belief network with a greedy layer-wise pre-training strategy, which allowed the transformation of previously learned representations of one task to be used in training of another task of the same input domain (LeCun, Bengio, and Hinton 2015; Goodfellow, Bengio, and Courville 2016, p. 19).

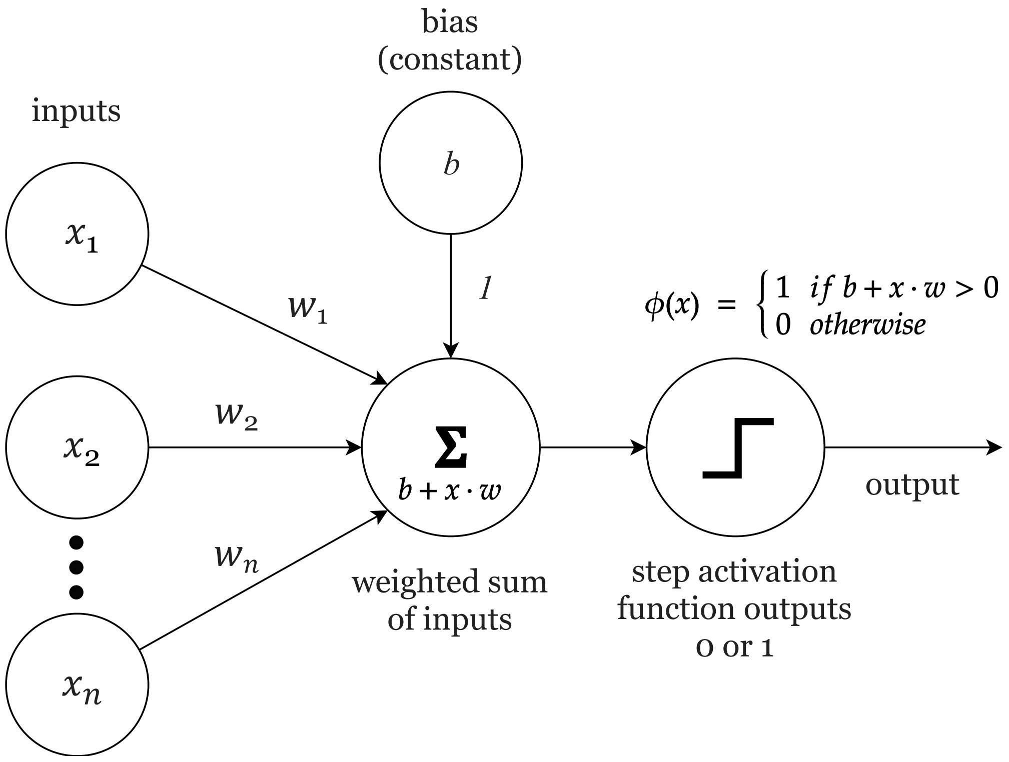


Figure 3. Simple perceptron.

The intention to understand biological learning via artificial neurons and networks led to the widely used term artificial neural networks along with the term deep learning. Current knowledge of the brain (and learning) and artificial intelligence, has benefited from the research of neural networks and deep learning. Deep learning-based models are not meant to be realistic models of the brain, but models inspired by biological functions of learning containing parts that have no relations to biological processes. (Goodfellow, Bengio, and Courville 2016, p. 15–17).

2.5 Network types and structures

In what follows, the descriptions of the network types and the structures shall be presented according to the textbooks of Goodfellow, Bengio, and Courville (2016) and Charniak (2018) if not cited otherwise.

A deep learning model is a multilayered network of nodes, where the input information flows through the network of hidden layers of nodes until reaching the output node(s) of the network (see Figure 4). Multilayered networks of nodes, where information moves only forward through the evaluating functions, or nodes, are called deep feedforward networks, feedforward neural networks, or multilayer perceptrons. Feedforward network is a conceptual base for many different types of neural networks, like recurrent neural networks where, in addition to feedforward, the outputs of the network are fed back forming feedback connections. A third common type of neural network is a convolutional neural network. The outputs of layer nodes are not fed to all inputs of the next layer, but only to the some via a convolutional layer. The convolutional layer contains operations like convolution, activation functions, and a pooling function modifying the output to be passed to the next layer.

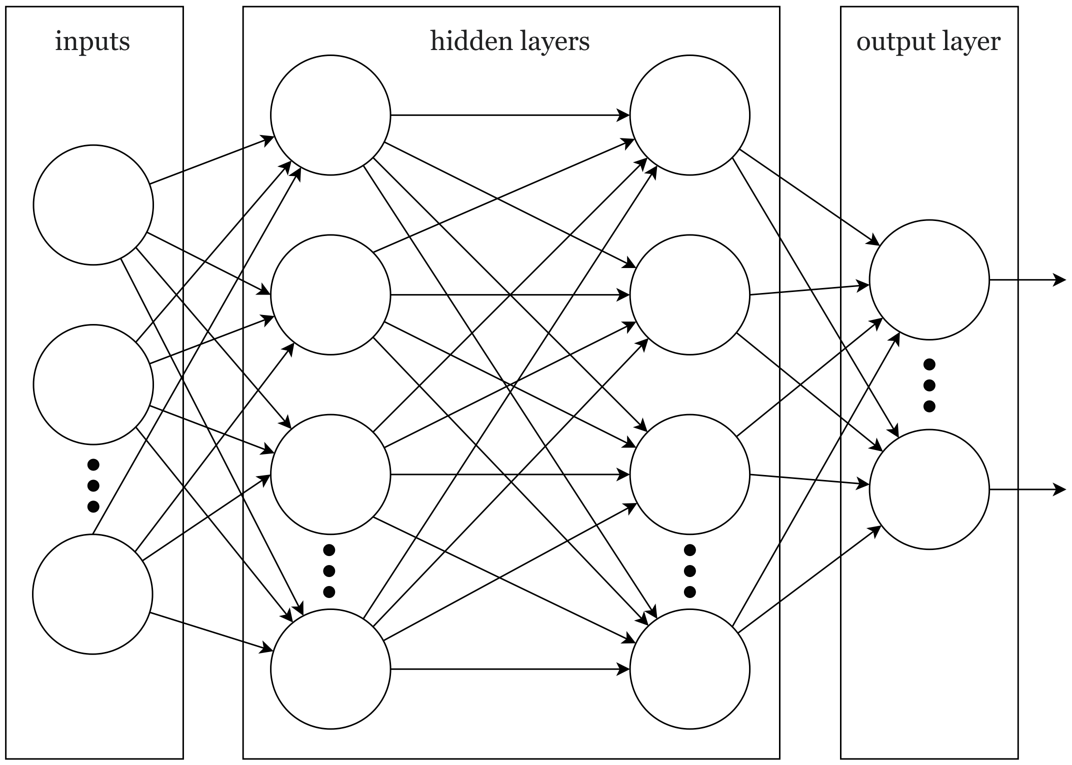


Figure 4. Feedforward neural network.

2.5.1 Feedforward neural networks

Understanding the feedforward neural network is essential for understanding deep learning and other types of neural networks. As previously mentioned, the feedforward neural networks consist of forward-connected nodes of different layers, as seen in Figure 4. The first layer contains inputs for the network. The second layer of the network is a hidden layer of nodes. Inputs of the networks are connected to each node of the first hidden layer. A hidden layer node sums weighted values of each input and the bias constant and outputs the value through the activation function which transforms the final output of the node (see Figure 5). Usually, the outputs of the hidden layer nodes are fed to the next hidden layer, and finally to the output layer, which contains one or many nodes with a similar node structure.

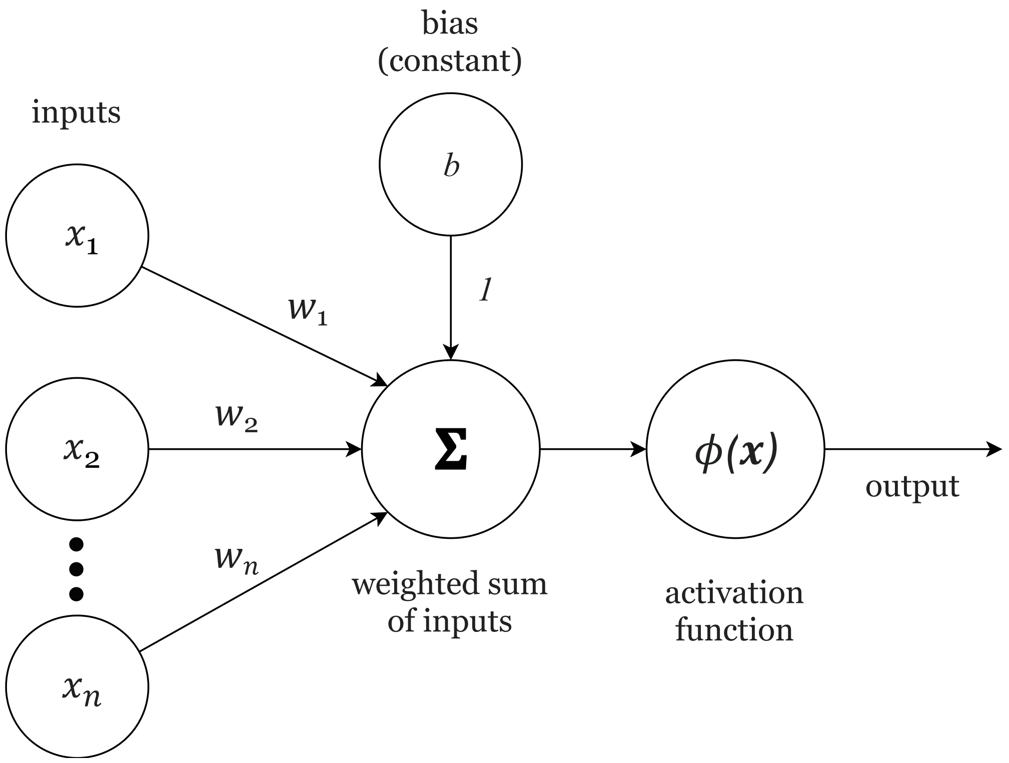


Figure 5. Single node structure in hidden layer and output layer.

The weight values and bias values are often initialized to small random values. A key part of a network and its nodes is a choice of activation functions. Many different types of activation functions are used. Perhaps one of the most common is a rectified linear unit (or ReLU, see Figure 6) that outputs 0 if the input value is below zero and outputs linearly for the positive input values. This linearity makes them easy to be optimized. Other common functions are logistic sigmoid and hyperbolic tangent activation functions, which are now less used than ReLU. Sigmoidal functions saturate quickly with large positive and low negative values. A sigmoid function is also sensitive when the value is near 0, which can make the learning very hard for the neural network.

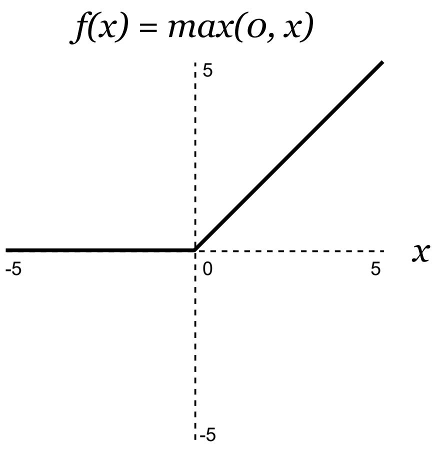


Figure 6. ReLU activation function.

Neural network "learns" through finding a desired local minimum of a gradient during the training by gradient descent optimization algorithm. To measure the progress of training, a suitable cost function (or loss function) is selected to indicate how well the network predicts the given test dataset. The goal of training the model is to minimize the output of the cost function, loss, via adjusting the weights and biases of the network by each iteration. Modern models are commonly trained by using maximum likelihood, which is achieved by using cross-entropy. Cross-entropy loss measures differences (cross-entropy) between two different distributions, for example between distribution derived from the training set (the true target distribution) and the probabilistic distribution of the model (an estimation of the target distribution). Minimizing the loss function can be visualized by finding the local or global minimum of a gradient by descending until the loss converges to a desirable local minimum.

The most important and common gradient descent algorithm is stochastic gradient descent. Stochastic gradient descent is efficient when training datasets are very large. In stochastic gradient descent, the gradient is an expectation estimated with a small set of samples, a minibatch, that is used to calculate gradient descent for a single iteration. During training, the whole dataset is iterated over and over multiple times; an iteration of the whole dataset is called an epoch. Calculating gradient descent reveals slope with the current parameters and points to the direction in which the values of the parameters should be adjusted to achieve the desired results. A situation where the gradient values are low and it is hard to know suitable parameters to adjust weights is called a vanishing gradient problem.

The level of changes in parameters can be controlled via learning rate. A large learning rate allows exploration of the gradients more broadly without falling into suboptimal local minima so easily, which is an issue with a small learning rate. A large learning rate can also speed up the process of finding local minima because calculating each gradient descent is time-consuming. The performance of calculating gradients in feedforward neural networks can be improved by using an algorithm called backpropagation, in which the output of loss function is fed backwards through the model and then used to calculate gradients.

2.5.2 Convolutional neural networks

Convolutional neural networks (see Figure 7) are usually used for purposes where the data is in a grid-like topology, like time-series data or images. To understand convolutional networks, it is best to explain basic concepts through the use of pixel image data as inputs of the network. A convolutional neural network input image is processed by a stack of convolutional layers which uses a convolutional kernel (or convolutional filter) to filter a subset of image pixels to the next hidden layer nodes. A common convolutional layer consists of three stages: 1) convolutions are used to produce linear activations, 2) non-linear activation functions are used to transform output, 3) a pooling function is used to modify the final output. Outputs of stages 1 and 2 are combined in Figure 7.

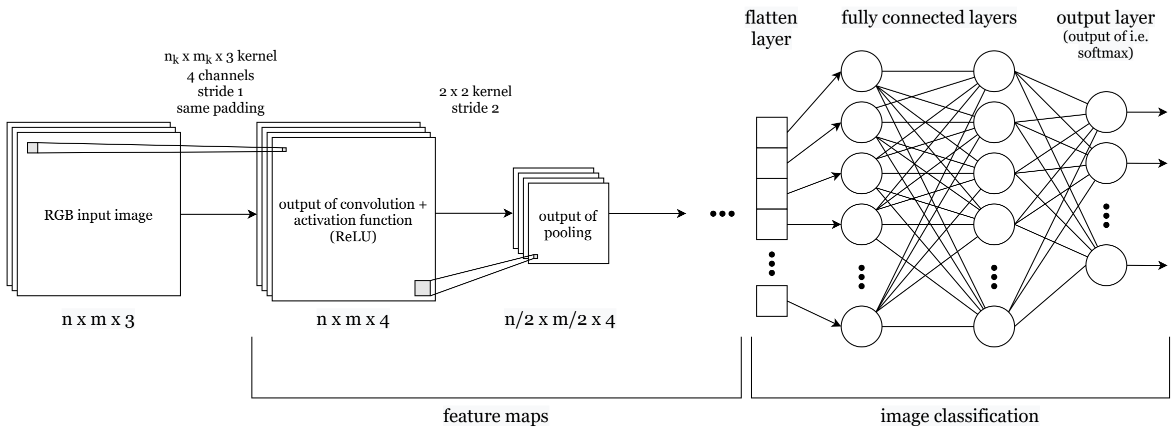


Figure 7. Convolutional neural network image classification.

In convolutional neural networks, there are several key components. Pooling algorithms, like max-pooling, where the maximum value of neighboring pixels is selected for output, help the network to become resilient to small differences in the input data. Different to the first layer after input, the input data of the hidden layers and the kernels are usually tensors, which are multidimensional arrays. The receptive field describes how many previous layer nodes account for the hidden layer's node value. The depth of the layer is defined by the number of channels (or kernels). Usually, the kernel is smaller than the original input image. Effectively this is sparse connectivity, where every input node is not connected to every hidden layer node, but only to some.

In convolution and pooling, the kernel is slid through every pixel of the image, or every nth pixel of the image, which is defined by a stride value. Padding can be used to control the size of the output of the convolution or pooling, and it affects how border values are calculated. Same padding means that made-up values (usually zeros) are added around the input so that the output size is the same as the original input size with a stride of 1. Valid padding means that only true input values are used. This always leads to a smaller output size compared to input size.

Like feedforward networks, the convolutional neural networks learn connected node weights and biases, but these are shared by all nodes of the layer. This means all nodes of the layer detect the same features, but from different parts of the input image; this is called parameter sharing.

For typical classification tasks, the output of the last layer is usually reshaped to flatten out the dimensionality to use it with typical classifiers familiar from regular feedforward networks. The learned features are typically fed to one or many fully connected layers and then to the classification output layer, where a softmax function (for instance) is used to calculate the probability of each class with the current input. A fully connected layer has all of its nodes connected to all nodes of the previous layer.

Often for an accurate image classification model, a large image dataset like ImageNet is used to train a base model based on deep networks like VGG or ResNet. DeepLabCut also uses these datasets and base models.

2.5.3 ImageNet

ImageNet (Deng et al. 2009) is a large-scale image dataset that contains millions (continuously increasing) of hand-annotated images to be used in machine learning tasks. ImageNet Large Scale Visual Recognition Challenge (or ILSVRC) (Russakovsky et al. 2015) is a yearly challenge to benchmark new methods on object category classification and detection, and it has been run annually since 2010. ImageNet is often used to train or pre-train different network structures (for example He et al. 2016; Ren et al. 2015; Pishchulin et al. 2016; Insafutdinov et al. 2016; Simonyan and Zisserman 2014) to achieve state-of-the-art results in classification and detection tasks.

2.5.4 VGG

VGG (Simonyan and Zisserman 2014) is a very deep (11-19 layers) convolutional neural network for image recognition tasks, which outperformed previous methods in the ImageNet Challenge 2014. The VGG network in Table 1 contains a sequential stack of convolutional layers separated by max-pooling layers at the front of each stack. The last convolution layer is followed by the last max-pooling layer, which is followed by a stack of three fully connected layers, ending at the softmax output layer. Convolution layer filters use a very small receptive field of 3x3 with 1 pixel stride while using 1 pixel padding to retain the spatial resolution of the input image. Max-pooling uses a 2x2 kernel with 2 pixel stride. The first two fully connected layers contain 4096 channels each, and the last layer performs a 1000-way classification containing 1 channel for each class. Rectified Linear Unit (or ReLU) is used as an activation function for the hidden layers. Many new very deep networks have emerged after VGG, like ResNet.

input
[conv 3x3, 64] x 2
2x2 maxpool, stride 2
[conv 3x3, 128] x 2
2x2 maxpool, stride 2
[conv 3x3, 256] x 4
2x2 maxpool, stride 2
[conv 3x3, 512] x 4
2x2 maxpool, stride 2
[conv 3x3, 512] x 4
2x2 maxpool, stride 2
4096-d fully connected
4096-d fully connected
1000-d fully connected
softmax output

Table 1. 19-layer VGG architecture.

2.5.5 ResNet

Deeper and deeper networks have always been hard to train. Residual neural network (ResNet) (He et al. 2016) was developed for image recognition tasks to address the issue, where the accuracy of the model degrades significantly when the depth of the network increases. He et al. (2016) reasoned that it is easier to optimize the residuals than the original mappings. They invented residual learning building blocks that contained two or three weight layers each followed by ReLU activation function, and the residual output is added with the input of a first weight layer via a shortcut connection (or identity mapping) before the last ReLU function. See "bottleneck" building block structure used for ResNet-50 in Figure 8. In the "bottleneck" image, the input depth is 256 and the depth is reduced to 64 via 1x1 convolutions and lastly, 1x1 convolution is used to increase depth back to 256. 1x1 convolution steps are added to shortcut connections where depth increases at the start of the each conv bank.

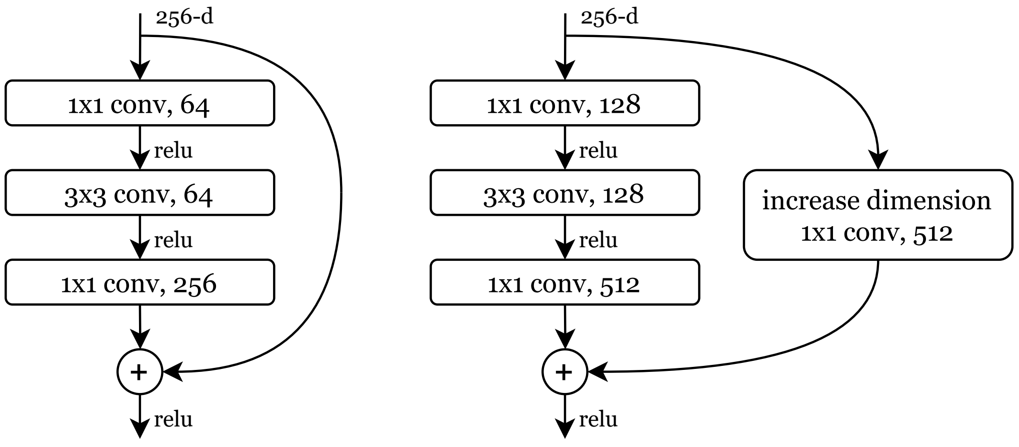


Figure 8. ResNet "bottleneck" structure.

ResNet got its inspiration from the VGG and the weight layers mostly have 3x3 or 3x3 and 1x1 convolutions (He et al. 2016). With the 1x1 convolutions, the width and the height are the same, but it can be used to change spatial dimensionality or channels so to speak. He et al. (2016) presented networks of different sizes from 18-layers up to 152-layers, which have slightly different residual blocks. ResNet-50 architecture presented in Table 2. In addition to ResNet architecture, He et al. (2016) used in their implementation a batch normalization after each convolution and before activation functions. Batch normalization is used to improve network learning performance and quality (Ioffe and Szegedy 2015).

layer name	output size	ResNet-50 (50-layers)
conv1	112x112	7x7, 64, stride 2
conv2_x	56x56	3x3 max pool, stride 2 [(1x1, 64), (3x3, 64), (1x1, 256)] x 3
conv3_x	28x28	[(1x1, 128), (3x3, 128), (1x1, 512)] x 4
conv4_x	14x14	[(1x1, 256), (3x3, 256), (1x1, 1024)] x 6
conv5_x	7x7	[(1x1, 512), (3x3, 512), (1x1, 2048)] x 3
	1x1	average pool 1000-d fully connected, softmax

Table 2. 50-layer ResNet architecture. Feature map size is downsampled by a factor of two (stride 2) in conv3_1, conv4_1 and conv5_1. Depth doubles from previous conv_x bank input every time the feature map size is halved.

Using weights of a ResNet pre-trained with the ImageNet dataset can be very useful, especially with small training data, to achieve accurate results in image detection and classification tasks as it is shown in the DeeperCut (and DeepLabCut), for example. Such use of pre-trained models is called transfer learning, which is a common practice in building neural network models (see Torrey and Shavlik 2010).

2.6 From DeepCut to DeeperCut and DeepLabCut

Perhaps the most valuable achievement of DeepLabCut was to utilize a state-of-the-art neural network method of a multi-person pose estimation to usable body part detection toolbox to use in real life research projects. Mathis et al. (2018) devised a feature detector part of the DeeperCut multi-person pose estimation model.

DeeperCut evolved from the firstly developed DeepCut (Pishchulin et al. 2016) method. Pishchulin et al. (2016) developed a body part detector called Dense-CNN to use in their multi-person pose estimator. They build a fully convolutional network around VGG to compute part probability scoremaps. The fully convolutional network is achieved by converting the fully connected layers to upsampling deconvolutional layers (or more accurately transposed convolution), which allows classification networks to output classification scoremaps with one-to-one correspondence to the input image pixels (Long, Shelhamer, and Darrell 2015; Dumoulin and Visin 2016). Scoremap describes the probabilities of every pixel belonging to some of the classification categories.

To target more precise body part localization, they used a hole algorithm to achieve a stride of 8 pixels instead of 32 pixels stride without modification to the fully convolutional VGG net. Input image heights were scaled to 320 pixels. Original VGG softmax detection layer was substituted with a better performing sigmoid activation function and cross entropy loss function. Pishchulin et al. (2016) introduced a location refinement method to improve precision via adding a fully connected location refinement layer after VGG's second fully connected layer named FC7 and using relative offsets of the scoremap locations and the ground truths as targets. In addition to location refinement, researchers used detected part location to regress relative positions of every other body part to improve the performance of the training.

Part detector accuracy was improved in DeeperCut from DeepCut by changing from VGG to fully convolutional ResNet base network (Insafutdinov et al. 2016). DeeperCut uses location refinement that is a predicted offset of the part location. Due to memory constraints, the hole algorithm was also replaced with a hybrid method to achieve 8 pixels stride. The final classification layer and averaging pooling layer were removed from the ResNet, then the stride of the first convolution layer of the conv5 was changed from 2 pixels to 1 pixel for keeping the original size and dilations were added to 3x3 convolutions of the conv5 for retaining receptive field size. After conv5, upsampling deconvolution layers were added, and the final output was combined with the output of the conv3. According to Insafutdinov et al. (2016) depth of the ResNet allows large receptive fields that improve accuracy by incorporating the context of the body part. To further improve the ResNet, researchers introduced modifications via intermediate supervision. Body part loss layers inside the conv4 were added. Both DeepCut and DeeperCut body part detectors used ImageNet pre-trained models for initialization. Both models use a stochastic gradient descent optimizer.

DeepLabCut uses (see Figure 9) Tensorflow implementation of the DeeperCut part detector created by one of the original authors of the DeeperCut. Implementation differs slightly from the original paper. For example, intermediate supervision is disabled on default, and a momentum algorithm is used with the stochastic gradient descent optimizer. Intermediate supervision was originally used only with ResNet-101 in Mathis et al. (2018), but not with the ResNet-50. According to one of the authors of DeeperCut[1], the implemented intermediate supervision is designed for the ResNet-101. For the ResNet-50 the intermediate supervision is not needed. Implementation code of the DeeperCut reveals that the location refinement layer uses Huber loss on default, and alternatively mean squared error, if Huber loss is not enabled.

DeepLabCut uses a spatial density map of probabilities, a scoremap, output of the DeeperCut part detector with the combination of location refinement offset. Scoremap describes probabilities for each body part in every pixel location of the evaluated frame, and the machine labeled part location is the point with the highest probability (or confidence) with location refinement offset taken into account (Mathis et al. 2018). Originally the DeepLabCut does

1. `https://github.com/eldar/pose-tensorflow/issues/1`

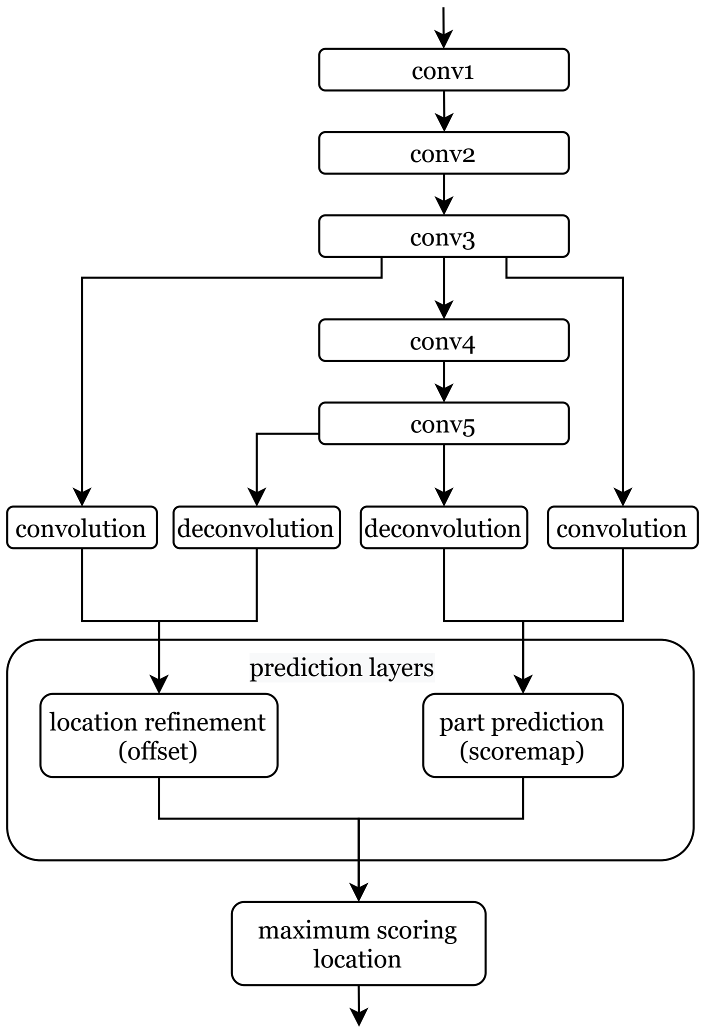


Figure 9. DeepLabCut ResNet-50. Single animal and no intermediate supervision.

not introduce any structural changes to the feature detector already developed in the DeeperCut, but uses its output for different purposes. Later Mathis and Mathis (2020) added support for different types of backbone networks, but in this Master's thesis, only the originally presented methods are tested.

3 Neural network model training

3.1 Dataset

Dataset for this thesis consisted of 21 videos of open field type experiments. The videos were hand-picked from the archives of previous experiments containing only a single white Sprague Dawley rat. Selected videos were from seven different experiments with varied open field experiment setups. The videos were recorded with different and unknown video cameras with varied frames per second, resolution, encoding, focus, and sharpness. A total of 18 videos were selected to the training dataset and 3 videos were selected to the validation dataset to test models' accuracy with a novel dataset.

Datasets were created with the DeepLabCut frame extraction feature, which supports clustering images based on visual appearance and predefined image count, along with other parameters. From 13 videos a maximum of 20 frames and from 8 videos a maximum of 120 frames were extracted with k-means clustering. Extraction resulted in a total of 1215 images, 959 frames for the training dataset, and 256 frames for the novel validation dataset.

All of the 1215 images were annotated with the GUI tool included in the DeepLabCut toolbox. There were nine body parts to annotate: body center, body left, body right, eye left, eye right, neck center, snout, tail stem, tail tip (see Figure 10). All visible body parts were annotated. With some body parts that were obstructed by objects like a preamplifier battery, landmark locations were estimated by eye. Body parts without clear landmark features were annotated broadly in relation to other body parts and the pose of the animal overall.

Since the used DeepLabCut version 2.1.7 did not support images without any annotations (images without visible body parts), the images without any annotations were removed. After removing the images, the sizes of the final datasets were 869 images for the training dataset and 217 images for the validation dataset.

DeepLabCut splits training and test datasets from the annotated images in a ratio defined in the project configuration file via parameter 'TrainingFraction'. Default value for the parameter is 0.95 (95 %). DeepLabCut shuffled 869 images of the training dataset images and 825

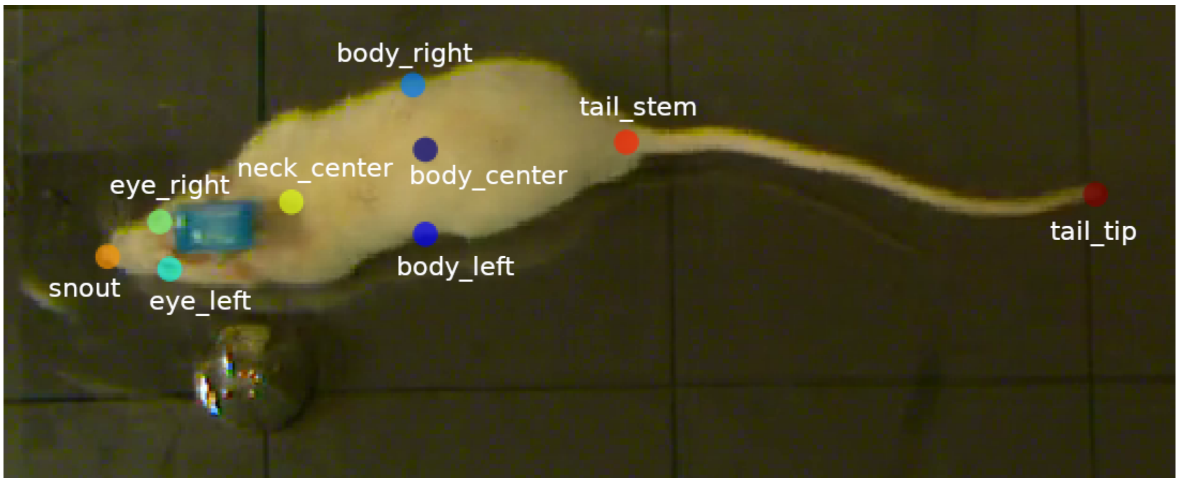


Figure 10. Annotated body parts.

images were selected to the final training set and 44 were selected to the training test set. For the novel validation dataset, all of the 217 images were used. Dataset details are listed in Appendix A.

3.2 Training

At first, the neural network model (Model D50) was trained with the default hyperparameters of the DeepLabCut for the ResNet-V1-50. After initial training, step sizes were changed and the model retrained from scratch (Model C50) to see if the initial Model D50 accuracy and convergence speed can be improved.

In DeepLabCut, the step sizes are passed to Tensorflow's MomentumOptimizer (stochastic gradient descent optimizer with momentum) to adjust the learning rate on each iteration. Momentum accelerates stochastic gradient descent in the right path and smooths steep surfaces in neighboring dimensions, thus decreasing the time needed for training (Ruder 2016). A third model (Model 101) was trained with Model C50 parameters, but with the ResNet-V1-101. Used parameters are released in the GitHub repository[1].

Training of a single model included a total of 930 000 iterations and lasted around 16–18

1. `https://github.com/lauritk/gradu2-dlc-projects/tree/main/model_configs`

hours with the ResNet-V1-50 and 20–24 hours with the ResNet-V1-101. The system used for training was Intel Core i7-4790k 4.4@4.7 GHz processor, 4 x 8 GB (32 GB) DDR3 2400 MHz memory, Samsung 970 Pro 1 TB NVME solid state hard drive, and Nvidia GTX 1080 Ti 11 GB model graphics card. The operating system was Ubuntu 19.10 and later 20.04. Latest proprietary NVIDIA linux drivers were used with CUDA 10. DeepLabCut 2.1.7 uses Tensorflow 1.13.1.

3.3 Results

To validate the accuracies of the models, they are tested against the training dataset, training test data, and a truly novel dataset. To test the accuracy between human-labeled body part coordinates and the model-detected coordinates, DeepLabCut uses mean Euclidean distances (Nath et al. 2019), instead of the root mean squared error (RMSE) as stated in the original Nature paper Mathis et al. (2018). The following Equation 3.1 can be obtained from the original code[2]. In the implementation, n is the number of items with existing values. In other words, the ’not a number’ -values are omitted. The ’predicted’ and ’actual’ values are coordinate points in pixels.

$$Error = \frac{1}{n}\sum_{i=1}^{n} ||predicted_i - actual_i|| \tag{3.1}$$

Confidence values are used to evaluate detection correctness. P-cutoff is a threshold value for DeepLabCut’s prediction likelihood (or confidence) (Nath et al. 2019). The higher the likelihood, the more confident the model is about the prediction. The 60 % confidence limit is taken from DeepLabCut defaults, but it is still fairly low.

Model D50 (Figure 11) with the default hyperparameters seems to achieve decent accuracy with moderately low iterations < 100 000. Due to the high learning rate, the accuracy fluctuated plenty after stabilizing till the next lower learning rate step. Model C50 (Figure 12) with the modified parameters also achieved similar or slightly better accuracy around the same training iteration. Model C101 (Figure 13) with the modified parameters and ResNet-

2. `https://github.com/DeepLabCut/DeepLabCut/blob/v2.1.7/deeplabcut/pose_estimation_tensorflow/evaluate.py`

V1-101 base model did similarly to previous models but did not improve accuracy compared to models that took less time to train. With Model C101 the first snapshot novel dataset accuracy was cut out of the plot for better comparability. The novel dataset error without the p-cutoff was 70.15 px and with 0.6 p-cutoff it was 92.64 px for the 10 000 iterations snapshot model.

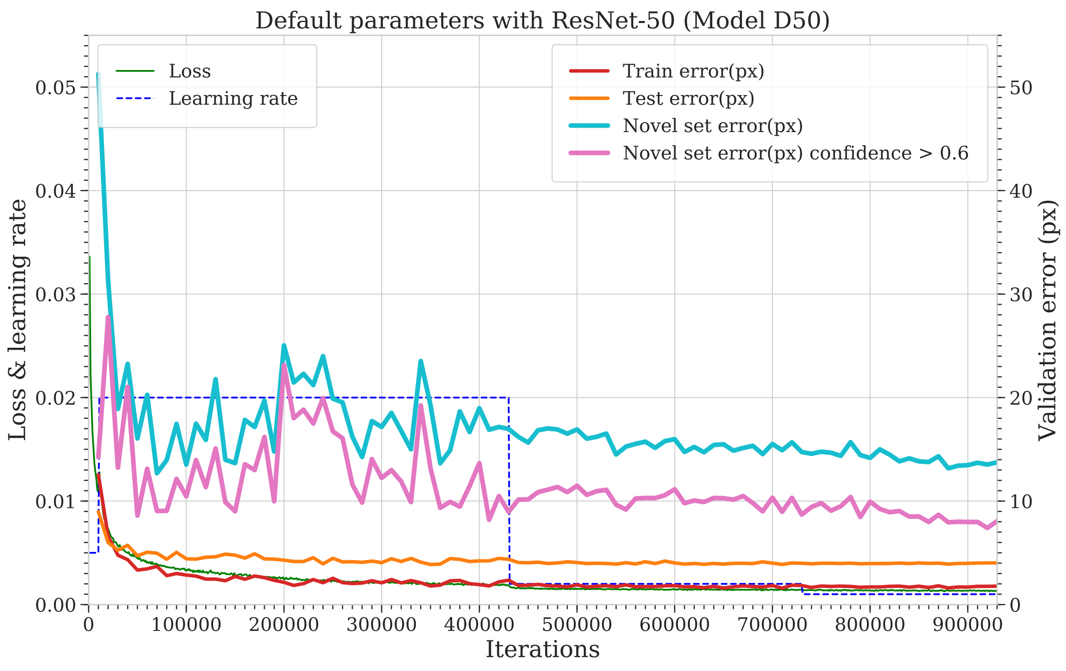


Figure 11. Model trained with the default parameters and ResNet-50 base.

All three models did fairly well with the novel dataset. The lowest error on a novel dataset with Model D50 was at the 70 000 iterations mark with 12.69 px error and with 0.6 p-cutoff 9.04 px. The lowest error with 0.6 p-cutoff was 7.4 px, but the overall error was 13.53 px at the 920 000 iterations. Model C50 results were more uniform and the lowest overall error with the novel dataset was 11.06 px and with 0.6 p-cutoff 8.85 px at the 520 000 iterations. The lowest error with 0.6 p-cutoff was 7.5 px with overall error of 12.47 px at the 340 000 iterations. Model C101's best novel dataset accuracy was 12.2 px and with 0.6 p-cutoff 7.89 px at the 820 000 iterations. The lowest error with 0.6 p-cutoff was 7.69 px with an overall error of 13.01 px at the 630 000 iterations.

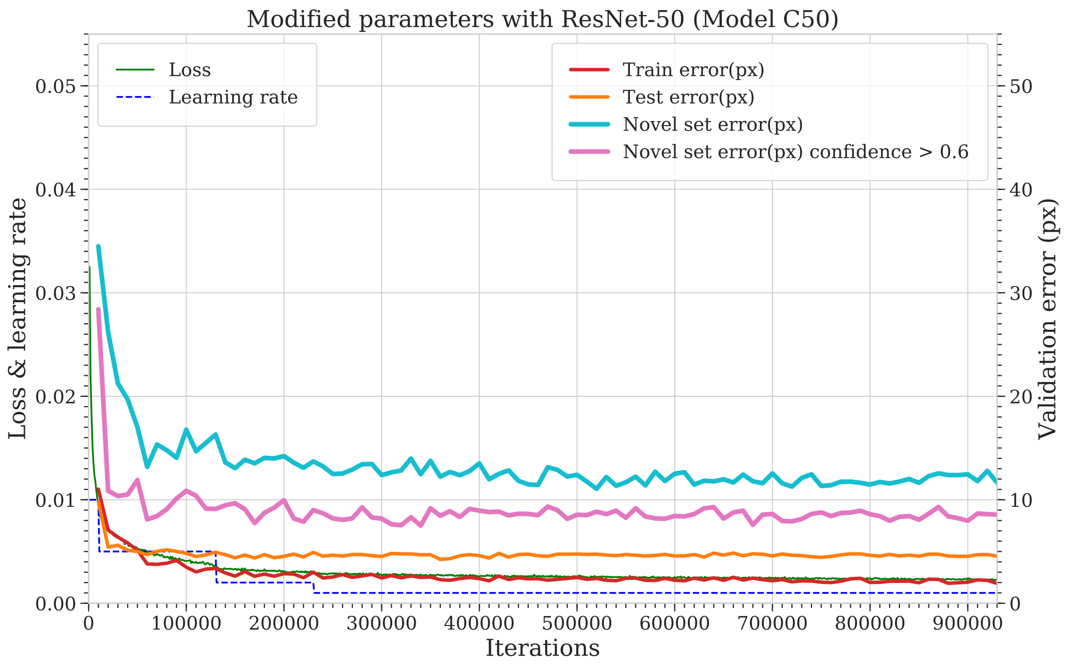


Figure 12. Model trained with the custom parameters and ResNet-50 base.

Naturally, all the models did pretty well with the training test set, which frames are taken from the same videos as the training set frames by shuffling the annotated frames and excluding test set frames from the training. The lowest error on a test dataset with Model D50 was at the 710 000 iterations mark with 3.88 px error and with 0.6 p-cutoff 3.87 px. The lowest error with 0.6 p-cutoff was 3.82 px and the overall error was 3.87 px at the 350 000 iterations. With Model C50 the lowest overall error with the training test dataset was 4.24 px and with 0.6 p-cutoff 4.09 px at the 360 000 iterations. The lowest error with 0.6 p-cutoff was 3.96 px with an overall error of 4.47 px at the 220 000 iterations. The Model C101's best training test dataset accuracy was 4.33 px and with 0.6 p-cutoff 4.1 px at the 510 000 iterations. The lowest error with 0.6 p-cutoff was 4.09 px with overall error of 4.41 px at the 220 000 iterations.

It is to be noted, that the errors in pixels are not relative but absolute to the frame size, therefore training dataset variable frame sizes affects error size differently. Also, the detection accuracy of different body parts could vary. To address these issues, novel dataset accuracy

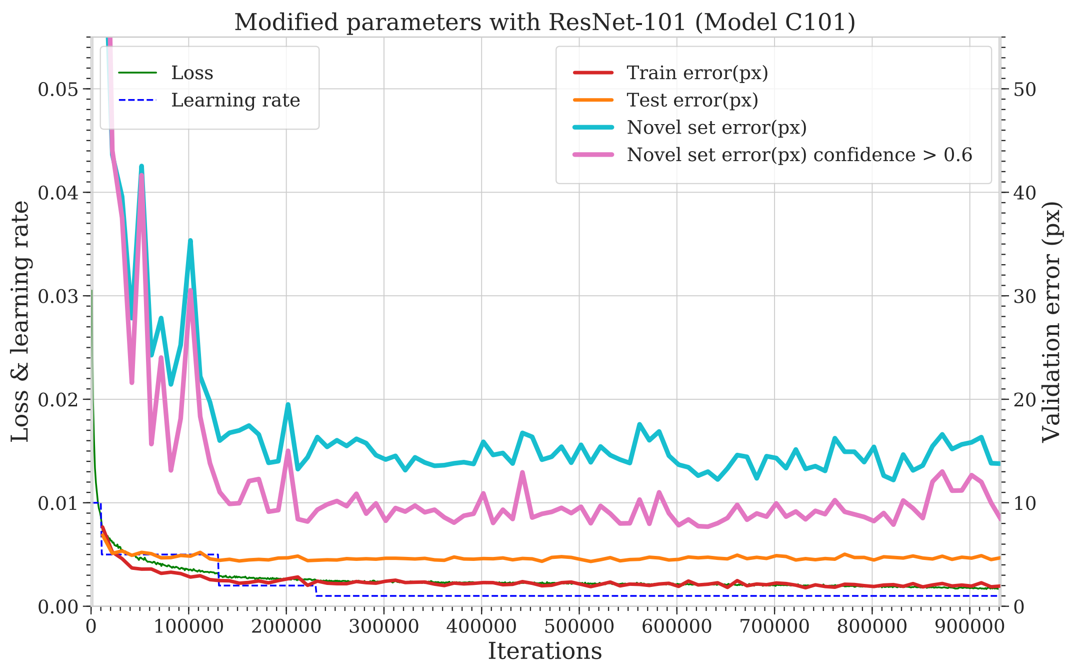


Figure 13. Model trained with the custom parameters and ResNet-101 base.

was explored by per body part (see Table 3) and also by per body part and per video (see Table 4). The model chosen for this analysis is the most accurate iteration of Model C50 with the novel set, which is at the 520 000 iterations mark.

n = 217	body center	body left	body right	eye left	eye right
all	8.46 (214)	11.34 (213)	10.80 (213)	5.83 (168)	4.63 (168)
p-cutoff 60 %	8.38 (199)	10.15 (194)	10.38 (183)	5.73 (162)	4.63 (162)
n = 217	**neck center**	**snout**	**tail stem**	**tail tip**	**mean**
all	8.58 (215)	7.42 (214)	8.96 (210)	32.40 (200)	11.06
p-cutoff 60 %	8.55 (210)	7.28 (212)	5.91 (196)	18.48 (178)	8.85

Table 3. Validation dataset accuracy in pixels per body part.

The resolution in video A14 is 960 x 540 px. It is from the same experiment (but separate video or trial) as 13 other videos in the training set. The novel dataset contains 15 frames from video A14. The resolution in video G1 is 1280 x 720 px and there are no same experiment recordings in the training set. Video G1 is a truly novel video. The novel dataset

	Video	A14 (n = 15)	G1 (n = 84)	F2 (n = 118)
body center	all	2.25 (15)	8.30 (84)	9.38 (115)
	p-cutoff 60 %	2.25 (15)	8.24 (82)	9.40 (102)
body left	all	2.55 (15)	13.04 (83)	11.25 (115)
	p-cutoff 60 %	2.55 (15)	10.74 (76)	10.81 (103)
body right	all	3.11 (15)	10.87 (84)	11.77 (114)
	p-cutoff 60 %	3.11 (15)	10.69 (79)	11.33 (89)
eye left	all	2.72 (15)	6.59 (54)	5.90 (99)
	p-cutoff 60 %	2.72 (15)	6.33 (48)	5.90 (99)
eye right	all	2.69 (15)	4.23 (57)	5.17 (96)
	p-cutoff 60 %	2.69 (15)	4.17 (51)	5.17 (96)
neck center	all	4.51 (15)	8.08 (83)	9.47 (117)
	p-cutoff 60 %	4.51 (15)	8.00 (82)	9.48 (113)
snout	all	2.73 (15)	8.09 (81)	7.55 (118)
	p-cutoff 60 %	2.73 (15)	7.91 (80)	7.44 (117)
tail stem	all	2.59 (15)	10.28 (79)	8.89 (116)
	p-cutoff 60 %	2.59 (15)	8.68 (76)	4.38 (105)
tail tip	all	19.85 (15)	70.09 (69)	11.60 (116)
	p-cutoff 60 %	2.36 (6)	43.33 (63)	5.00 (109)
mean	all	4.78	15.23	9.12
	p-cutoff 60 %	2.87	11.82	7.62

Table 4. Validation dataset accuracy in pixels per video and body part.

contains 118 frames from video G1. Video F2 is one of the two high-quality experiment recordings. Its resolution is 1192 x 1010 px and the novel dataset contains 84 frames from it. Video F2 is from the same experiment and session as video F1, but with a different type of an open field setting. See Appendix A for a detailed listing and description of used videos.

On average, video G1 was the toughest to accurately analyze with the C50 model. Even though it had the largest resolution, but no counterparts in the training dataset. Video A14 was most accurate, but it did have 13 counterparts in the training dataset. Body parts from video F2 were detected fairly accurately, and it did have only one similar counterpart in the training dataset. Eyes were the most accurately detected and tail tips were the hardest to identify. Eyes were also clearest and easiest to label. It seems that high-quality recordings are good candidates for detecting body parts accurately when there are some similar frames in the training data. There were 107 frames from video F1 in the training dataset. Truly novel video detection was almost as accurate, but the inaccurate detection of tail tip shifted the overall accuracy. Other than that, the model seems feasible in research use without additional training data or training.

The datasets and the DeepLabCut project files will be released in the author's GitHub repository[3]. The trained models can be inquired from the author.

3. `https://github.com/lauritk/gradu2-dlc-projects.git`

4 Using tracking in a practical application

During the development of the animal body parts tracking model, the open field experiment analysis software Rodent Open Field Tracker (ROFT, Figure 14) was also developed partly in the context of this Master's thesis and another course. The purpose of the open field analysis software was to have a research-ready application to be used in real-life experiments like in context-object discrimination tasks presented earlier.

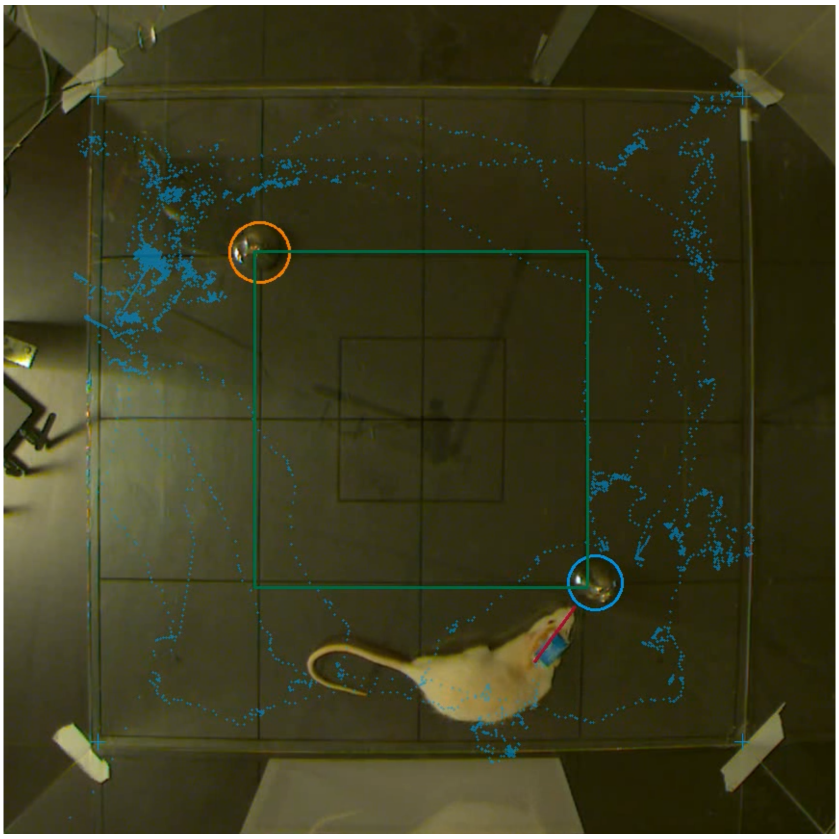


Figure 14. Rodent Open Field Tracker analysis visualized.

ROFT can be used to measure the time and distance moved by an animal in the arena, the moving velocity of the animal, the time spent in different sections or sectors of the arena, and time spent exploring objects in the arena. The exploration of the objects is detected by the distance of the animal from the object, and the angle of the head towards the object. Time spent at each sector is defined by a criterion of how many body parts are inside the sector. All the used body parts, criteria, and thresholds are defined in the project configuration file.

After conducting the experiments, the first phase of analyzing the data is to detect each body part locations with the DeepLabCut model presented here. After obtaining all body part locations for all frames of the video, the data is imported to ROFT via configuration. The user must load the video file into ROFT by running video configurer with the video path as a parameter. Guided configurer helps to define outside dimensions of the arena, arena shape, corners, sectors, and objects in the arena (parameters in Appendix B). The user must set project-wide parameters that define used thresholds and body parts that are used to calculate key figures (see Appendix C). The project-wide settings can be set by copying and modifying the provided project configuration template file. After initial configuration, it is possible to modify configurations via configuration editor. Screenshot of the configurer is in Figure 15 and screenshot of the editor is in Figure 16.

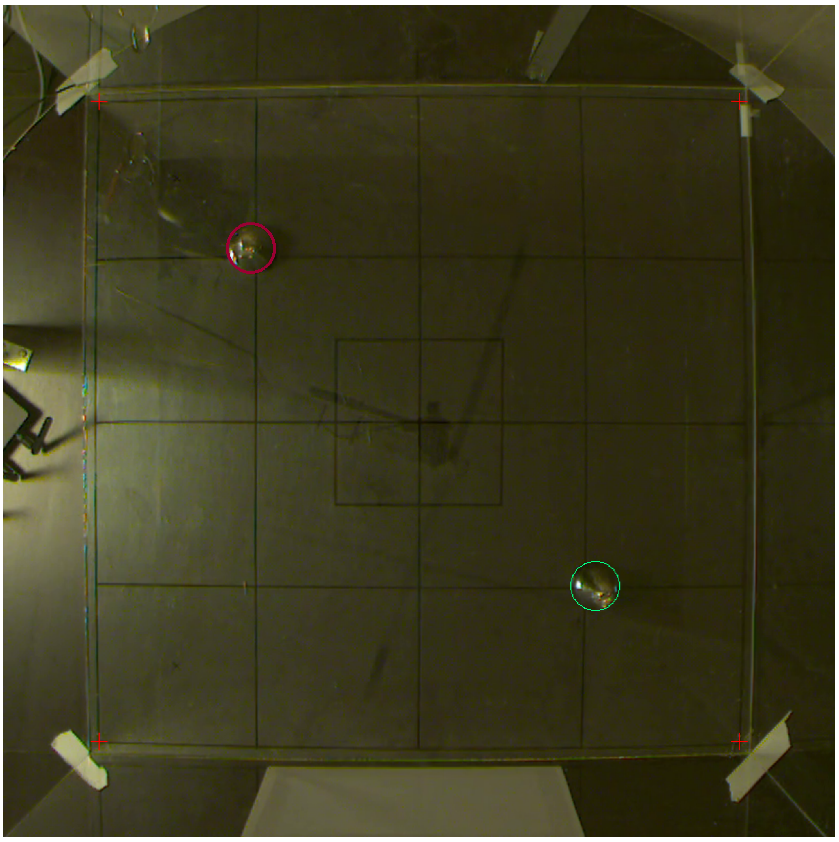
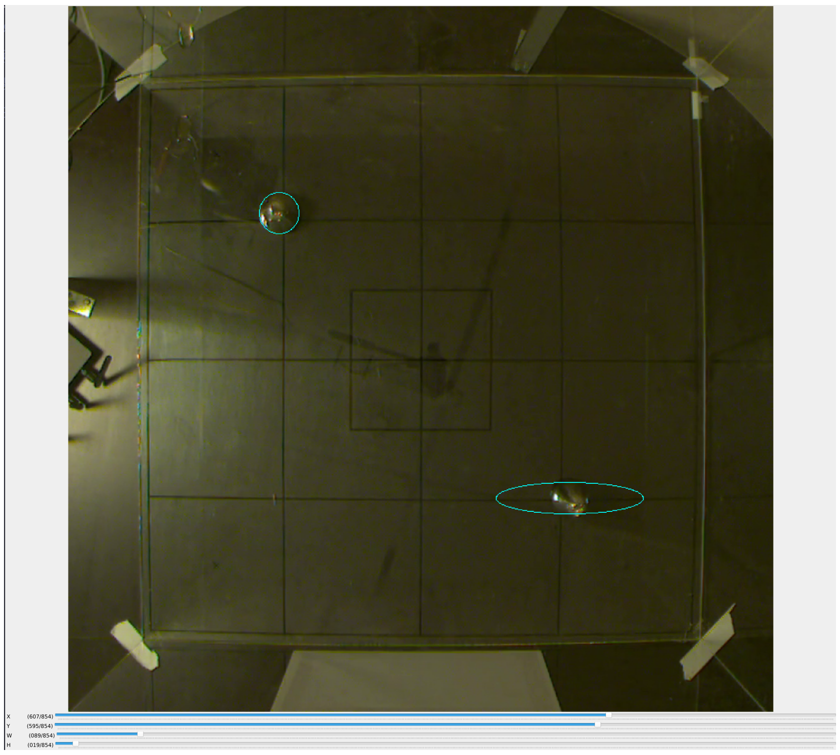


Figure 15. Configuring objects in the configuration step.

After configuration, the video can be analyzed. In the analysis, rudimentary perspective correction is applied to correct body part locations. Arena dimensions in centimeters are



```
X    (607/854)
Y    (595/854)
W    (089/854)
H    (019/854)
```

Figure 16. Editing object shape in the configuration editor.

used to translate pixel values to centimeters. Frames are converted to time in seconds. The frame rate of the video defines the temporal accuracy and the resolution spatial accuracy. See Appendix D for a full list of extracted figures. After analysis, the data can be plotted against the video to see if the configurations work as expected. Example of plotted data in Figure 14.

The program is written in Python 3 and it uses libraries like OpenCV, Numpy, Pandas, and Numexpr. The Rodent Open Field Tracker will be released in the author's GitHub repository[1] under the GNU General Public License v3 (see GPL 2007).

1. `https://github.com/lauritk/open-field-tracker.git`

5 Discussion

A state-of-the-art method was tested for tracking animals in a neuroscience experiment. Practical use of tracking data demonstrated via a self-tailored experiment analysis program, where a user can extract context-object task-related key figures from obtained tracking data of animal body part coordinates.

DeepLabCut is suitable for tracking animals in an open field experiment (RQ1). As seen in the results, the accuracy is near to human-labeled body part accuracy. Similar results can be seen in fairly new research by Sturman et al. (2020) comparing DeepLabCut with commercial tracking software. DeepLabCut even outperformed the commercial programs. The authors developed the analysis further and used skeletal representations to assess ethologically relevant behaviors on par with human accuracy. The authors also released the R programming language script collection DLCAnalyzer to analyze DeepLabCut tracking results similar to a custom program presented here but containing much more features.

DeepLabCut is a moderately easy-to-use toolbox that can be used via a graphical user interface, and with the help of step by step guide lowers the bar to start using it for researchers not familiar with programming. DeepLabCut provides tools for annotating, forming dataset, training models, evaluating models and running inferences on the input data.

DeepLabCut is also actively developed and new methods are introduced to make it more accurate and suitable for broader use. As an example, DeepLabCut-Live! (Kane et al. 2020) was recently released for a real-time application. DeepLabCut-Live! is also integrated with Bonsai (see Lopes et al. 2015) and Autopilot (see Saunders and Wehr 2019) via plugins. The integration allows closed-loop experiments to be built that uses real-time tracking of DeepLabCut-Live!. In addition to real-time usage, DeepLabCut can now be used for multi-animal tracking when assessing the social behavior of animals (Lauer et al. 2021).

Using DeepLabCut makes experimenting and analyzing data more robust and straightforward than any previous method used before. There is no need for special environments or markings, nor tedious manual labor, except annotating training data, adjusting training parameters, training, and evaluating output quality. As shown here, the result can be used for extracting meaningful figures when assessing animal behavior.

DeepLabCut is accurate for tracking animal body parts, but the accuracy depends on the quality of the training data and the input material (RQ2). Overall accuracy ranged from a few pixels to 10–15 pixels, excluding the tail tip which seemed to be most difficult to track. Results might have been a bit different with a different shuffled set of the training and evaluation data, but due to the practical nature of this thesis, time constraints, and adequate accuracy, we settled for the given results. In practical use, the trained model is often optimal to only some acceptable degree. The final dataset was quite small and took only a few days to annotate. The dataset contained videos of varying quality to get a broader sense of the accuracy and see the effect on the results. DeepLabCut works well with low-quality videos when there are many frames from similar or same experiments and cameras. The models worked reasonably well with high-quality and high-resolution videos, even though there was less training data from the high-quality recordings. The trained model worked well even with a truly novel video. Only tail tip accuracy differed greatly from the videos that had similar experiment data in the training dataset. More data would bring the tail tip accuracy closer to the others. The tail tip was hard to annotate because the tail moves fast and causes motion blur at low shutter speeds. As a rule of thumb it is advisable to record experiments in well-lit (normal or IR-lights) environments, and use high resolution and sensitive cameras with good optics with little distortion. Also, it is preferable to use cameras whose frame rates and other parameters are adjustable and constant, along with high-quality video encoding. It is often uncertain how consumer-grade cameras adjust parameters etc. which is crucial when measuring e.g. movement and behavior in relation to time and space.

To improve the model even further it would be beneficial to collect labeled data from research projects and train models further to make it more accurate and general for different environments and animals. Augmentation was not used here, but as DeepLabCut supports the imgaug library (Mathis et al. 2020), it is certainly recommended to be tested. Image augmentation means a way to alternate images of the dataset in different ways to improve accuracy and generalization (Perez and Wang 2017).

It is possible to use tracking data to extract meaningful key figures with a developed program (RQ3). The developed program was used to measure the movement velocity of animals, distance and time spent around objects and different sectors of the area. Resulted coordinates from the DeepLabCut always contain distortion from the optics and angle of the camera. Also, the distance of the camera from the target affects the size of the region of interest in the video frame, which is why pixel values are not comparable between videos. In the program, a simple perspective correction is applied to the video frames and pixels-values are converted to centimeters. After corrections and conversions, extracted values can be used to compare between sessions and animals. Frames are also converted to timestamps. It is important that the camera and the encoding capture every frame to obtain reliable temporal measures. To obtain more precise temporal information higher frame rate can be used. Usually recordings vary between 20–30 frames per second. Higher frame rates can result in dropped frames due to higher processing demands, which has to be taken into account when choosing a suitable experiment hardware.

The developed program is a proof-of-concept and is by no means complete. One crucial part is to support more complex shapes like polygons to approximate object shapes and enable the use of more complex arenas like radial arm mazes (see Lensu et al. 2019). Due to time constraints, the objects were only approximated by rectangles, circles, and ellipsoids. Also, perspective correction works best with a rectangle area, where there are four clear corners to adjust the perspective and to skew area dimensions even. Perspective correction is also rudimentary and it does not take into account the optics used. To obtain more precise values it is advisable to implement proper camera lens correction to tackle distortion like radial, barrel, and pincushion distortions (see Montabone et al. 2009). To adopt wider use it would be wise to rewrite the user interface along with the analysis code for better maintainability.

As technology progressed rapidly after 2018, there are now multiple different methods based on neural networks for tracking animals in experiments. Perhaps the most notable counterparts for DeepLabCut are SLEAP (Pereira et al. 2020), which is a successor of now deprecated LEAP (Pereira et al. 2019), DeepPoseKit (Graving et al. 2019), DeepBehavior (Arac et al. 2019) and APT: The Animal Part Tracker[1], which is a complete set of tools for animal tracking including several different tracking algorithms like for example DeepLabCut and developers' mixture density network-based algorithm. As tracking individual parts and ani-

1. https://kristinbranson.github.io/APT/

mal positions has already improved to great accuracy, the evolution of the methods is heading to a fully automated multi-animal social behavior assessment with 3D tracking, as can be seen from the progress of these methods (Ziegler, Sturman, and Bohacek 2021). Automatic social behavior assessment streamlines research, opens a whole new world of experimental designs, and enables new discoveries that no group can afford to overlook.

The value of this Master's thesis is to present how neural network-based tracking works, and how it can be utilized in a research project. Also, it provides a base for the research group to continue from. Neural network-based tools are new methods, in which many neuroscientists have little experience. Hopefully, this thesis spark interest to start using them in real-life experiments.

Bibliography

Aimone, James B, Yan Li, Star W Lee, Gregory D Clemenson, Wei Deng, and Fred H Gage. 2014. ”Regulation and function of adult neurogenesis: from genes to cognition”. *Physiological reviews* 94 (4): 991–1026.

Arac, Ahmet, Pingping Zhao, Bruce H Dobkin, S Thomas Carmichael, and Peyman Golshani. 2019. ”DeepBehavior: A deep learning toolbox for automated analysis of animal and human behavior imaging data”. *Frontiers in systems neuroscience* 13:20.

Burgess, Neil. 2014. ”The 2014 Nobel Prize in Physiology or Medicine: a spatial model for cognitive neuroscience”. *Neuron* 84 (6): 1120–1125.

Charniak, Eugene. 2018. *Introduction to Deep Learning.* MIT Press.

Denenberg, Victor H. 1969. ”Open-field behavior in the rat: What does it mean?” *Annals of the New York Academy of Sciences* 159 (3): 852–859.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ”Imagenet: A large-scale hierarchical image database”. In *2009 IEEE conference on computer vision and pattern recognition,* 248–255. Ieee.

Dumoulin, Vincent, and Francesco Visin. 2016. ”A guide to convolution arithmetic for deep learning”. *arXiv preprint arXiv:1603.07285.*

GPL. *GNU General Public License.* 2007. Version 3. Free Software Foundation. `http://www.gnu.org/licenses/gpl.html`.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* The MIT Press.

Gould, Todd D., David T. Dao, and Colleen E. Kovacsics. 2009. ”The Open Field Test”. In *Mood and Anxiety Related Phenotypes in Mice: Characterization Using Behavioral Tests,* edited by Todd D. Gould, 1–20. Totowa, NJ: Humana Press.

Graving, Jacob M, Daniel Chae, Hemal Naik, Liang Li, Benjamin Koger, Blair R Costelloe, and Iain D Couzin. 2019. ”DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning”. *Elife* 8.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. ”Deep Residual Learning for Image Recognition”. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh. 2006. ”A fast learning algorithm for deep belief nets”. *Neural computation* 18 (7): 1527–1554.

Insafutdinov, Eldar, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. 2016. ”DeeperCut: A Deeper, Stronger, and Faster Multi-person Pose Estimation Model”. In *Computer Vision – ECCV 2016,* 34–50.

Ioffe, Sergey, and Christian Szegedy. 2015. ”Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37,* 448–456. ICML’15.

Kane, Gary A, Gonçalo Lopes, Jonny L Saunders, Alexander Mathis, and Mackenzie W Mathis. 2020. ”Real-time, low-latency closed-loop feedback using markerless posture tracking”. *Elife* 9:e61909.

Kantola, Lauri, Tiina Pirttimäki, and Miriam S Nokia. 2017. ”Aikuisiän neurogeneesi hippokampuksessa mahdollistaa joustavan toiminnan”. *Psykologia* 52 (6): 436–456.

Lauer, Jessy, Mu Zhou, Shaokai Ye, William Menegas, Tanmay Nath, Mohammed Mostafizur Rahman, Valentina Di Santo, Daniel Soberanes, Guoping Feng, Venkatesh N Murthy, et al. 2021. ”Multi-animal pose estimation and tracking with DeepLabCut”. *bioRxiv.*

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. ”Deep learning”. *Nature* 521 (7553): 436–444.

Lensu, Sanna, Tomi Waselius, Markku Penttonen, and Miriam S Nokia. 2019. ”Dentate spikes and learning: disrupting hippocampal function during memory consolidation can improve pattern separation”. *Journal of neurophysiology* 121 (1): 131–139.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. 2015. ”Fully convolutional networks for semantic segmentation”. In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 3431–3440.

Lopes, Gonçalo, Niccolò Bonacchi, João Frazão, Joana P. Neto, Bassam V. Atallah, Sofia Soares, Luís Moreira, et al. 2015. ”Bonsai: an event-based framework for processing and controlling data streams”. *Frontiers in neuroinformatics* 9:7.

Mathis, Alexander, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. 2018. ”DeepLabCut: markerless pose estimation of user-defined body parts with deep learning”. *Nature neuroscience* 21 (9): 1281–1289.

Mathis, Alexander, Steffen Schneider, Jessy Lauer, and Mackenzie Weygandt Mathis. 2020. ”A Primer on Motion Capture with Deep Learning: Principles, Pitfalls, and Perspectives”. *Neuron* 108 (1): 44–65.

Mathis, Mackenzie Weygandt, and Alexander Mathis. 2020. ”Deep learning tools for the measurement of animal behavior in neuroscience”. *Current opinion in neurobiology* 60:1–11.

Montabone, Sebastian, Frank Pohlmann, Brian MacDonald, Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, et al., editors. 2009. ”Distortion Correction”. In *Beginning Digital Image Processing: Using Free Tools for Photographers,* 185–204. Berkeley, CA: Apress.

Nath, Tanmay, Alexander Mathis, An Chi Chen, Amir Patel, Matthias Bethge, and Mackenzie Weygandt Mathis. 2019. ”Using DeepLabCut for 3D markerless pose estimation across species and behaviors”. *Nature protocols* 14 (7): 2152–2176.

Noldus, Lucas PJJ, Andrew J Spink, and Ruud AJ Tegelenbosch. 2001. ”EthoVision: a versatile video tracking system for automation of behavioral experiments”. *Behavior Research Methods, Instruments, & Computers* 33 (3): 398–414.

Nolen-Hoeksema, S, Rita L Atkinson, Ernest R Hilgard, Barbara. Fredrickson, Geoffrey R Loftus, and Willem Wagenaar. 2009. *Atkinson & Hilgard's Introduction to Psychology.* Atkinson and Hilgard's Introduction to Psychology. Cengage Learning.

Pereira, Talmo D, Diego E Aldarondo, Lindsay Willmore, Mikhail Kislin, Samuel S-H Wang, Mala Murthy, and Joshua W Shaevitz. 2019. "Fast animal pose estimation using deep neural networks". *Nature methods* 16 (1): 117–125.

Pereira, Talmo D, Nathaniel Tabris, Junyu Li, Shruthi Ravindranath, Eleni S Papadoyannis, Z Yan Wang, David M Turner, Grace McKenzie-Smith, Sarah D Kocher, Annegret Lea Falkner, et al. 2020. "SLEAP: Multi-animal pose tracking". *BioRxiv.*

Perez, Luis, and Jason Wang. 2017. "The effectiveness of data augmentation in image classification using deep learning". *arXiv preprint arXiv:1712.04621.*

Pishchulin, Leonid, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter V Gehler, and Bernt Schiele. 2016. "Deepcut: Joint subset partition and labeling for multi person pose estimation". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 4929–4937.

Ramsaran, Adam I, Sara R Westbrook, and Mark E Stanton. 2016. "Ontogeny of object-in-context recognition in the rat". *Behavioural brain research* 298:37–47.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2015. "Faster r-cnn: Towards real-time object detection with region proposal networks". *Advances in neural information processing systems* 28:91–99.

Rosenblatt, Frank. 1958. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65 (6): 386.

Ruder, Sebastian. 2016. "An overview of gradient descent optimization algorithms". *arXiv preprint arXiv:1609.04747.*

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. "Imagenet large scale visual recognition challenge". *International journal of computer vision* 115 (3): 211–252.

Saunders, Jonny L, and Michael Wehr. 2019. ”Autopilot: Automating behavioral experiments with lots of Raspberry Pis”. *bioRxiv:* 807693.

Simonyan, Karen, and Andrew Zisserman. 2014. ”Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556.*

Spink, AJ, RAJ Tegelenbosch, MOS Buma, and LPJJ Noldus. 2001. ”The EthoVision video tracking system—a tool for behavioral phenotyping of transgenic mice”. *Physiology & behavior* 73 (5): 731–744.

Sturman, Oliver, Lukas von Ziegler, Christa Schläppi, Furkan Akyol, Mattia Privitera, Daria Slominski, Christina Grimm, et al. 2020. ”Deep learning-based behavioral analysis reaches human accuracy and is capable of outperforming commercial solutions”. *Neuropsychopharmacology* 45 (11): 1942–1952.

Torrey, Lisa, and Jude Shavlik. 2010. ”Transfer Learning”. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques,* 242–264. IGI Global.

Ziegler, Lukas von, Oliver Sturman, and Johannes Bohacek. 2021. ”Big behavior: challenges and opportunities in a new era of deep behavior profiling”. *Neuropsychopharmacology* 46 (1): 33–44.

Appendices

A Dataset videos

File	Dataset	Annotated	Selected	Resolution	Label
2019-03-13 14-54-53	training	20	14	1280x720	A1
2019-03-13 15-03-01	training	20	16	1280x720	A2
2019-03-13 15-13-28	training	20	17	1280x720	A3
2019-03-13 15-20-38	training	20	19	1280x720	A4
2019-03-13 15-26-49	training	20	17	1280x720	A5
2019-03-13 15-33-18	training	20	15	1280x720	A6
2019-03-13 15-40-22	training	20	17	1280x720	A7
2019-03-13 17-05-35	training	20	17	1280x720	A8
2019-03-14 13-54-44	training	20	17	1920x1080	A10
2019-03-15 14-05-01	training	20	16	604x340	A11
2019-05-15 13-02-29	training	20	16	1280x720	A12
2019-05-15 14-15-16	training	20	14	960x540	A13

Continued on next page

Continued from previous page

File	Dataset	Annotated	Selected	Resolution	Label
COD_ exposure1B_ rats5to8_ RAT5	training	120	113	640x480	B1
COD_ expo-sure_1A_ rats17and19_ RAT17	training	119	107	1280x720	C1
PreIrr PlaceTest_ rats17-20_ RAT17	training	120	107	640x360	D1
Rat25_27_ COD4_A_ test_RAT25	training	120	120	640x360	E1
Rat35_COD4_ testA_cut	training	120	120	720x400	E2
TIM18 SDM018_ ACAC_1_ alkuosa	training	120	107	1192x1010	F1

Continued on next page

Continued from previous page

File	Dataset	Annotated	Selected	Resolution	Label
2019-05-15 14-45-58	validation	19	15	960x540	A14
Rat36_34_ COD1-test_ A_RAT34	validation	119	118	1280x720	G1
TIM18 SDM018_ SEREX_ 2_alkuosa	validation	118	84	1192x1010	F2

Table 5. Details of the dataset videos.

B ROFT per video configuration

Per video configuration contains following parameters:

- **video_file** is the video filename.
- **video_frame_count** is the total frame count of the video.
- **video_fps** is the video's frames per second.
- **video_length_ms** is the video duration in ms.
- **video_original_size** is the original dimensions of the video.
- **video_corrected_size** is the dimensions of the video after perspective correction.
- **analysis_start_frame** is the frame where the analysis is to be started.
- **analysis_end_frame** is the end frame of the analysis.
- **experiment** is the experiment name.
- **experiment_phase** is the experiment phase or trial.
- **field_width** defines the open field width in cm.
- **field_height** defines the open field height in cm.
- **field_shape** defines the open field shape.
- **corners** is a list of coordinates of the corners of the open field.
- **objects** is a list of coordinates and parameters of the items.
- **sectors** is a list of coordinates and parameters of the sectors.
- **transformation_matrix** is a transformation matrix for the perspective correction.

C ROFT project-wide configuration

Project-wide configuration contains following parameters:

- **deeplabcut_name** is DeepLabCut-postfix in videos tracking result h5-file.
- **angle_th** defines the maximum angle that is counted as an exploration of an item.
- **distance_th1** is the minimum distance from the item's edge with the angle threshold that is counted as an exploration of an item.
- **distance_th2** is the minimum distance from the item's edge without the angle threshold that is counted as an exploration of an item.
- **save_parts** is a list of body parts loaded for the analysis.
- **angle_parts** is a list of body parts that are used for measuring the angle with the items and the animal, e.g. ['snout', 'neck_center'].
- **sector_parts** is a list of body parts that are taken into account in time spent in each sector.
- **sectors_criteria** defines how many body parts must be inside the sectors at least.
- **track_part** is the body part used only for drawing animal track.
- **base_part** is the body part used to calculate movement, distance and speed.
- **explore_part** is the body part used to calculate distance from the items edges.
- **likelihood_limit** defines the threshold limit of a tracked point to be taken into account. Values below the threshold are interpolated between closest good values.
- **still_limit** is a threshold value to detect if the animal is still. Limit is a distance moved between two frames in cm.
- **visit_gap** is a threshold value in frames for joining two close visits together if the gap is less than the threshold.
- **visit_length** is a threshold value in frames for joining two close visits together if the visit length is less than the threshold.

D ROFT extracted figures

Depending on the experiment, researchers are interested in many key figures that are derivable from the open field. The key figures extracted with the Rodent Open Field Tracker are listed below:

- **total_frames** is the video frame count in a selected time window (from trial start to end).
- **duration_s** is the duration in seconds in a selected time window.
- **items_explored** is the time in frames spent at each item.
- **items_visits** is a count of separate visits at each item.
- **items_visit_idx** is an index of each item visit. Zero is a trial start frame.
- **sectors_inside** is the time in frames spent at each sector.
- **sectors_visits** is a count of separate visits at each sector.
- **sectors_visit_idx** is an index of each sector visit. Zero is a trial start frame.
- **distance_cm** is a distance animal moved in cm in a selected time window.
- **frames_still** is a count of frames the animal is not moving.
- **raw_speed_cm_s** is a mean speed of an animal in cm/s in a selected time window.
- **corrected_speed_cm_s** is a corrected mean speed of an animal in cm/s in a selected time window when the animal is moving.