NTNU
Norwegian University of
Science and Technology

Automated Salamander Recognition Using Deep Neural Networks and Feature Extraction

Author(s)

Jørgen Bakløkken
Felix Schoeler
Hugo Nørholm

Bachelor of Science in Engineering - Computer Science
BIDAT39
Department of Computer Science
Norwegian University of Science and Technology,

20.05.2019

Supervisor Sony George
Marius Pedersen

Sammendrag av Bacheloroppgaven

Tittel:	**Automatisert salamander identifisering ved bruk av dype kunstige nevrale nettverk og feature extraction**
Dato:	20.05.2019
Deltakere:	Jørgen Bakløkken Felix Schoeler Hugo Nørholm
Veiledere:	Sony George Marius Pedersen
Oppdragsgiver:	Norsk institutt for naturforskning - NINA
Kontaktperson:	Børre Dervo
Nøkkelord:	Salamandere, Mønstergjenkjenning, Dyp læring, Nevrale nett, Maskinlæring
Antall sider:	49
Antall vedlegg:	3
Tilgjengelighet:	Åpen

Sammendrag:

Hensikten med dette prosjektet er å utvikle et automatisert verktøy og metode som kan identifisere salamandere ved å bruke deres individspesifikke bukmønstre. Vi har undersøkt en rekke med forskjellige eksisterende metoder for og oppnå dette. Mens de tradisjonelle metodene krever tidskrevende manuel retting av salamandrene, foreslår vi et system som gjør dette helautomatisk. Vi undersøker en rekke forskjellige metoder for å oppnå dette. I tillegg oppnådde vi en god gjenkjenningsrate. Den endelige versjonen bruker det foldbare nevrale nettverket ResNet50 og kubisk kurve interpolasjon for bukmønster lokalisering, og sammenlikner bildenes tette scale-invariant feature transform (DSIFT) for gjenkjenning.

Summary of Graduate Project

Title:	**Automated Salamander Recognition Using Deep Neural Networks and Feature Extraction**
Date:	20.05.2019
Authors:	Jørgen Bakløkken Felix Schoeler Hugo Nørholm
Supervisor:	Sony George Marius Pedersen
Employer:	Norsk institutt for naturforskning - NINA
Contact Person:	Børre Dervo
Keywords:	Salamanders, Pattern recognition, Deep learning, Neural networks, Machine learning, Feature extraction, Pose estimation
Pages:	49
Attachments:	3
Availability:	Open

Abstract:

The purpose of the project is to develop a automated tool and a method that can recognize salamanders by using their individual-specific body markings. We have investigated a number of different methods to accomplish this. While traditional methods require time intensive manual straightening of the salamanders, we propose a system that does this fully automatically. Additionally, we achieved a good recognition rate. The final implementation of the software uses the convolutional neural network ResNet50 and cubic curve interpolation for belly pattern localization, and compares the images dense scale-invariant feature transform (DSIFT) for recognition.

Contents

List of Figures

List of Tables

Preface

This thesis report was written by a group of three computer science students at Norwegian University of Science and Technology. We would like to thank NINA (Norwegian Institute for Nature Research) and Børre Dervo for the interesting and challenging task.

We would also like to thank our supervisors Sony George and Marius Pedersen for helping us with constructive feedback and their professional expertise and guidance during our project.

1 Introduction

1.1 Background and motivation

The goal for this project is to propose a solution which:

- Identifies salamanders based on pattern recognition with a high accuracy, comparable to existing software.
- Reduces the amount of manual work required to match salamanders by automating the workflow.

One of the fields of research at NINA (Norwegian Institute for Nature Research) is salamanders. Salamanders are a group of amphibian and are predators that hunt insects and earthworms. In Norway there are two different species of salamander, the great crested newt (Triturus crustatis) and the smooth newt (Lissotriton vulgaris). As the great crested newt (Triturus cristatus), has the status NT (Near threatened) in Norways national red list, it is crucial to know how it's population is developing. When measures are taken to increase the salamander population, the development of the population gives a good indicator for whether the actions had a positive impact.

NINA are trying to learn more about their life cycle, living area and population. And to get more knowledge they started a pit tagging project in 2010. This means that they have to catch the salamanders in a gentle way, inject a pit tag in the salamander and document this into their database. This pit tag is a small transponder contained in material that is harmless to the salamander [2], this transponder is the placed in the salamander and responds to the scanner by sending its id to the scanner allowing for identification of the animal. This is very time consuming for NINA and also stressful for the salamander that should be handled very carefully.

The great crested newt and smooth newt have body markings of small black patches that are individual-specific [3] (see Figure 1 and Figure 2). These individual-specific body markings can serve as a natural mark for capture–recapture models for estimating population demography. NINA would like a system or application that takes an input image of an salamander, check the pattern, determines if the salamander is already in the database.

The purpose of the project is to develop a tool and a method that can estimate the population demography for amphibian.

1.1.1 Capture-recapture

In order to estimate the population the capture-mark-recapture method is used. To calculate the development of the population we look at the ratio between the individuals that have been caught previously and the ones that have not. By using the Lincoln-Peterson method we

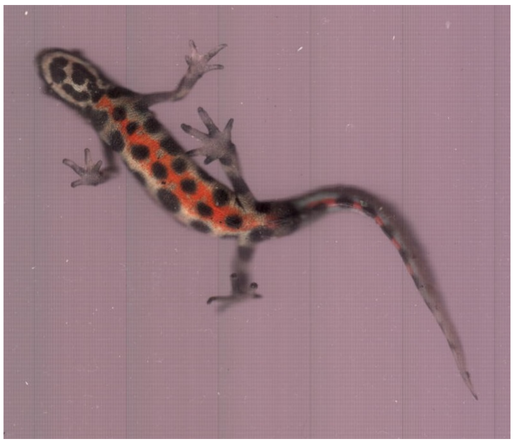


Figure 1: Example of salamander 1

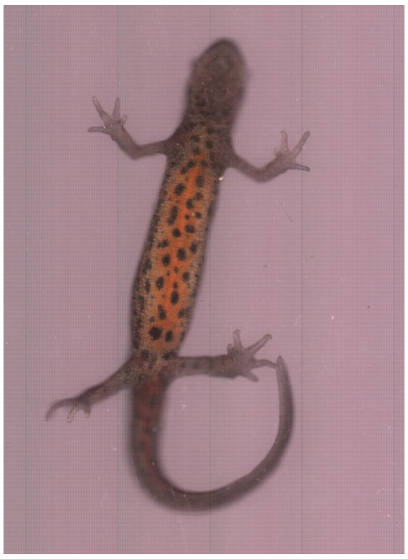


Figure 2: Example of salamander 2

can then estimate the population.

- N is the total number of salamanders in the population.
- n is the number of salamanders that were marked on the first visit.
- K is the number of salamanders that were captured on the second visit.
- k is the number of recaptured salamanders that were marked.

If we assume that the population is closed, meaning that no individuals die or emigrate between the visits, the ratios of the captured individuals in relation to the total population should be equal to the ratio between the individuals that have a mark on recapture and those that don't.

$$\frac{N}{n} = \frac{K}{k} \tag{1.1}$$

If we rearrange, we see that we can estimate the total population $\hat{N}$ using the following equation:

$$\hat{N} = \frac{Kn}{k} \tag{1.2}$$

1.1.2 Overview of existing software

In order to spark some ideas on how we were going to make the program we looked for programs that tried to achieve similar things. We found an article [3] which describes four different programs which attempts to aide with image identification of amphibians. The plan is to try to use these different programs on some of our images in order to see how they work and consider which of their features might be worth implementing in our project. We are not going to look too much into how well the programs work and instead use the results the article found since some of them are very old and the images we have are not very high quality.

The programs have two main methods of comparison feature-based and pixel based. I3s and Wild-ID both use a feature based methods which try to identify keypoints in the images and compare them to the other image's keypoint. While Aphis has both feature based and pixel based methods we are only going to look at its pixel based implementation. Pixel based matching is used by AmphIdent and Aphis, the method involves looking at the pixel similarity of the two images and uses this to find the matches.

I3s

The method for comparison used in this program involves you defining three homologous reference points, in the case of the salamander one could choose front left leg, front right leg and the beginning of the tail. It then identifies key points in the pattern e.g. the black dots on the salamander and calculates a similarity score based on how close these key points are to the key points from another image. For a more detailed look at the method look at [3]. This program was found at[4],

Wild-ID

This was probably the simplest of the programs to use as all you had to do was select the folder containing all the images, you then had to go through a list of potential matches for each image and confirm whether or not they actually were the same. A bit tedious, but it could be useful to show more than the highest match in case there are multiple high matches above a threshold. The program has no way of qualifying images for a match it simply ranks all the images by how similar they are to the image you are using. More about Wild-Id can be found at[5].

AmphIdent

The only non open source program on this list, it is also the one with the most features and support. There is even a plugin you can buy which directly relates to one of the species that we are working with. However all of the versions use the same basic algorithm, what changes is the algorithm it uses to binarize the images as different species have different colours which means they need different thresholds to decide if a pixel is black or white, therefore we only looked at the trial version. The way AmphIdent compares images is simply by adding up the absolute difference of each pixel. Additionally in order to improve robustness against small differences which might appear due to growth, camera faults or warping during the preprocessing, AmphIdent scales and translates one of the images many different ways and compares each of the resulting images and picks the best match for their comparison score. AmphIdent's store page where a free demo version is available[6].

Aphis

Aphis works similarly to AmphIdent, but it separates itself by identifying the area where the two images are most similar first before comparing just this area. This seems as it would require a lot less processing power than AmphIdent's compare the entire image approach, but it also sounds a little less accurate. Aphis also doesn't do the initial downscaling or the differently scaled comparisons.[7]

Conclusion

The main takeaway from looking at these programs is that no matter what kind of method of comparison we use, we need at least two pre-processing steps which are straightening and snipping out the belly pattern. We are going to test out both methods of comparison before and compare them to see which one we should use with our images. Another takeaway is that they all require human input in order to achieve the pre-processing steps that are required, we are going to attempt to automate these steps to reduce the need for human input as much as possible. A comparison between these softwares and our system can be found in Table 8. It is also worth mentioning that this article[3] was conducted by AmphIdent.

1.2 Our dataset

The dataset we received from NINA was a total of ~6700 images. The images were of the northern crested newt (Triturus crustatis) and the smooth newt (Lissotriton vulgaris). The resolution of these images is was 1240 x 1748 and they were provided in JPEG format. For the salamanders that were PIT tagged and recaptured, we also received data on which images are of the same salamander. All photographs were photographed using a passport-scanner.

Some of the images in this dataset were of low quality, including examples of:

- Partially or fully occluded belly pattern.
- Very dark images with low contrast, making the pattern unrecognizable.
- Salamanders were lying with a curved pose, some of them even in circles.
- Dirt and other noise added to the image.
- Varying background.
- The Salamander lying on one side of it's trunk.
- Incorrectly focused photos.
- Overexposed photos.

Because of the varying quality in the dataset, and not all images could be used for matching, we extracted parts of the dataset with acceptable quality for testing the different parts of the system as it is outlined in Section 3.1.4 and Section 3.2.

1.3 Tools and hardware

1.3.1 Hardware system

The hardware used for computations in this thesis is outlined in Table 1.

Component	Model
GPU	Geforce GTX 1060 6GB
CPU	Intel Core i5-2310@2.9 GHz
RAM	DDR3 8GB

Table 1: Hardware system used for testing in this thesis

1.3.2 Software tools

In this project we have used Python 3.6 as our programming language which is a popular programming language and allows for fast prototyping. We chose Python because it has a lot of good libraries, and is widely used, in the fields of machine learning and computer vision.

Python packages/libraries used in this project:

- Open Source Computer Vision Library - opencv-contrib-python==3.4.2.16
- Open Source Computer Vision Library - opencv-python==3.4.2.16
- Scientific computing package for python - numpy =1.16.1
- Scientific computing package for python - scipy =1.2.1
- 2D plotting package for python - matplotlib =3.0.3
- Image processing package for python - scikit-image =0.15.0
- Paralell computing package for python - dask =1.1.5
- Skeleton Network for python - https://github.com/yxdragon/sknw
- Complex network package for python - networkx==2.2
- JSON serialization module - jsonpickle==1.1
- Read and write HDF5 fileformat - h5py==2.9.0
- DeepLabCut - deeplabcut==2.0.6.3
- LEAP - `https://github.com/talmo/leap`

1.4 Report overview

This report includes some fundamental background in Chapter 2 that's necessary to read the whole report. Chapter 3 includes information and results about the methods we have decided to use or not to use in our finalized software. Figure 3 shows which algorithms and methods were used in the final software.

1.4.1 Project pipeline

Our project goal is to create a system that can identify salamanders based on pattern recognition with the high accuracy. If we achieve this, this will make it easier for NINA to identify salamanders. It will also reduce the amount of pit tags in salamanders.

Figure 3 shows our current methods. The techniques marked in red were tested and evaluated, but we decided not to use them. We found that the ones marked in blue were the best ones for our use case and they were used in the final proposed software.

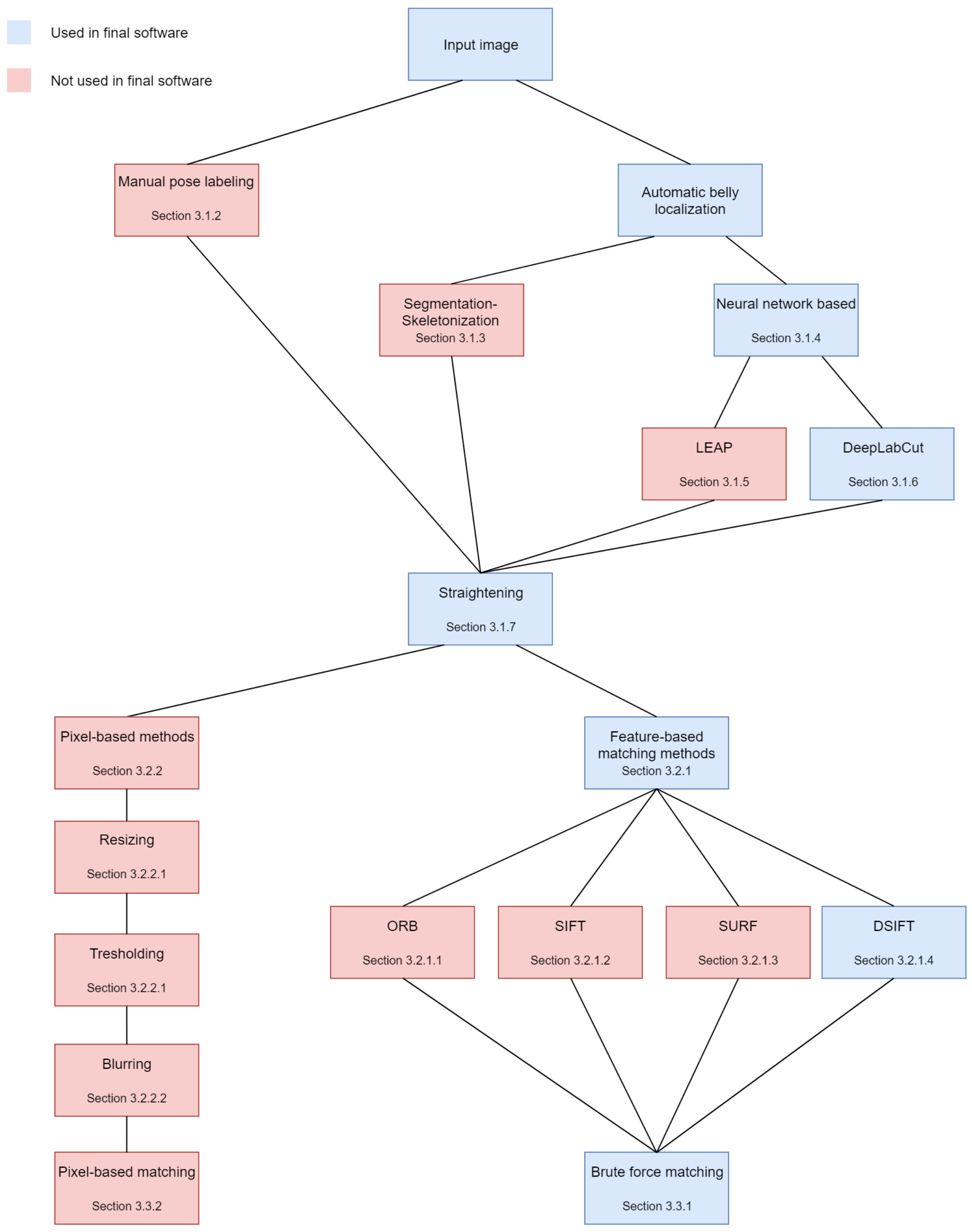


Figure 3: Chart showing the complete process from an input image, through all preprocessing steps until it is matched

2 Project fundamentals

2.1 Digital image processing

Figure 4 shows the fundamental steps in digital image processing. In this project we will use the representation and description section, which is also called feature extraction.

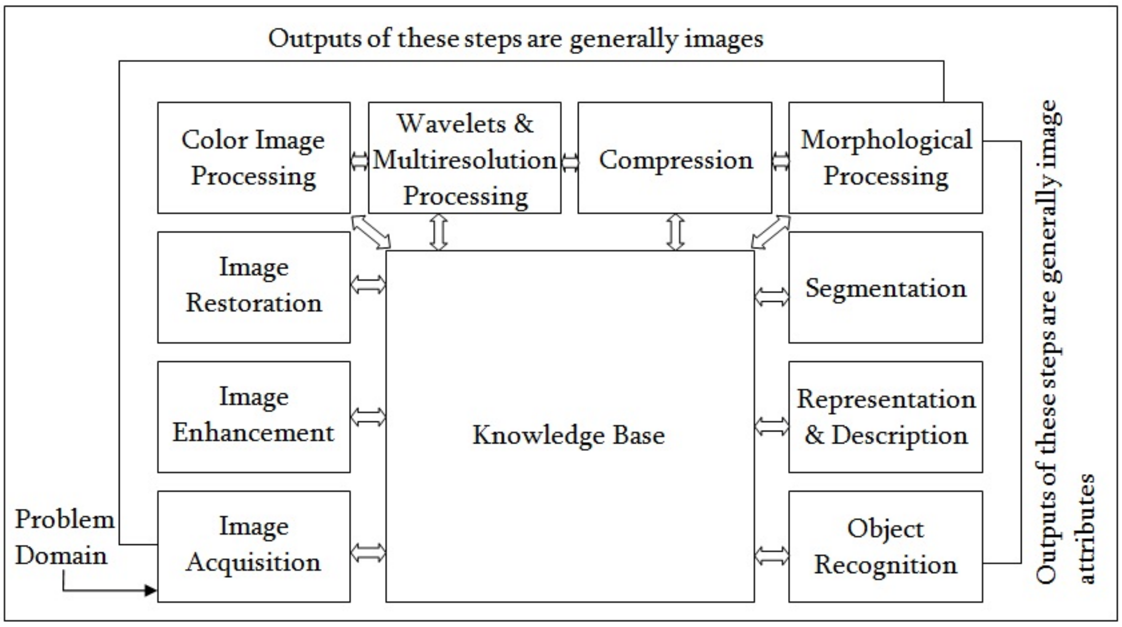


Figure 4: Fundamental steps in digital image processing [8]

More about digital imaging and digital image processing can be found in Digital Image Processing [9].

2.1.1 Feature extraction

The goal with feature extraction is to transform the original data to a data set which contains useful data that's unique to one image. There are two aspects of feature extraction: feature detection and feature description. Feature detection will decide if there's an image feature in a point of the image or not. Feature description will describe the image feature from feature detection and this data can be used to separate images from each other.

There's many different algorithms to use depending on what features you want from the image:

- Edge detection [10]
- Corner detection [11]
- Oriented FAST and rotated BRIEF (ORB) [12]
- Scale-invariant feature transfrom (SIFT) [13]

- Speeded up robust features (SURF) [14]
- Thresholding [15]
- Blurring [16]
- Hough transform [17]

These algorithms are the most popular ones for low-level and shape based features. In this project we tested the thresholding, blurring, ORB, SIFT and SURF algorithms.

2.1.1.1 Thresholding

Thresholding is a very simple method for image segmentation. Thresholding takes a grayscale or colour image and creates a binary image as seen in Figure 5. In addition we know that AmphIdent[6] uses binary images for their pixel-based matching algorithm.

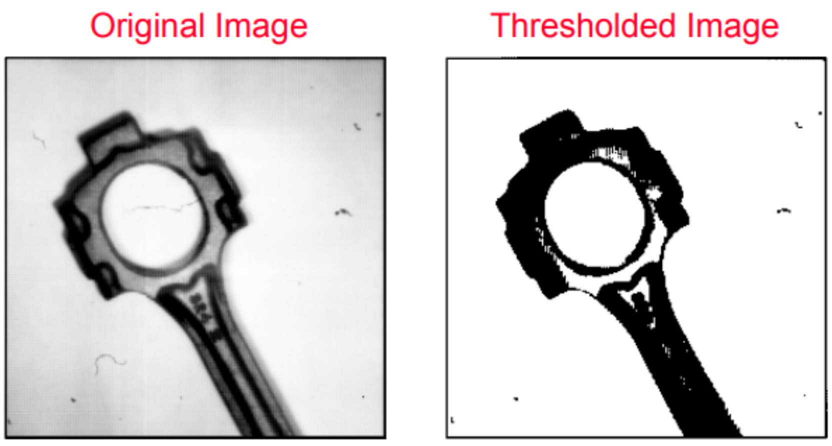


Figure 5: Simple thresholding example

- The simplest way of thresholding is to define the threshold as an arbitrary number and then changing each pixel to be black or white depending on whether they are brighter or darker then the threshold This method works quite well if you have the right threshold value, however when applying this to a variety of images of varying lightning creates a need for many different values as using a threshold value that works well on a bright image will turn a dark image become way too dark.

- Adaptive threshold decides on a threshold on a per pixel basis based on the value of the nearby pixels, the simplest one being the mean of the nearby pixels. This method works very well when trying to highlight lines and other this details as while they might not be too visible for us they are clearly darker or brighter than their surrounding pixels to the computer. However this method was not very useful for our purposes as we are trying to highlight bigger patterns which when using this method becomes fragmented as pixels in the pattern that are slightly brighter the surrounding pixels become white.

- Otsu's binarization works very similarly to simple thresholding however instead of using an arbitrary value for the threshold Otsu's method determines the threshold based on the histogram of the image. This method only works well with images that consist of two main colours also known as bimodal images. Our images of the pattern consist mainly of black and orange and should therefore be suited for this method. The reason Otsu's method requires bimodal images is because these images histograms have two distinct peaks around the image's two main colors and calculates the value between the two peaks and uses that to binarize the image.

2.1.1.2 Blurring

The main goal of blurring is to make each pixel more like the surrounding ones. This might be useful for us as a lot of images contain a lot of noise and blurring them might help if the noise becomes problematic.

- Gaussian blurring is a spin on the simplest method of doing blurring which is to make each pixel the average of itself plus its surrounding eight pixels. The Gaussian way of doing this is to use a box filter instead, this method is best when trying to remove Gaussian noise.

- Median blurring instead of using the average value uses the median value. This helps a lot with salt and pepper noise as this kind of noise deviates highly from the actual image and would therefore affect the average too much to be used accurately. Instead it uses the median value as this value is very likely not to be affected by extreme outliers.

- Bilateral filtering works like Gaussian filtering, however pixels that deviate too much from the center are disregarded in order to preserve edges, This type of blurring is best when you want to remove the texture of the items in the image while keeping edges clear as it basically only blurs within the edges.

2.1.2 ORB

Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector. ORB is based on the FAST keypoint detector and the visual descriptor BRIEF (Binary Robust Independent Elementary Features).

2.1.3 SIFT

Scale-invariant feature transform is a feature detection algorithm that is used to detect and describe local features in images. SIFT has 4 key steps:

1. Scale-space extrema detection

2. Keypoint localization
3. Orentation assignment
4. Keypoint descriptor

SIFT identifies points of interest (keypoints) using DoG (Difference of Gaussian Filtering) in the first two steps and then uses HOG (Histogram of Oriented Gradients) to describe these interest points to create a keypoint descriptor in the two last steps.

2.1.4 SURF

Speeded up robust features can be used for tasks such as object recognition, image registration and classification. SURF is built on SIFT and the same steps as SIFT but small details in each steps are different. SIFT is faster and also claimed to be more robust against different image transformations than SIFT.

2.2 Neural network fundamentals

2.2.1 Neurons

The layers in the neural networks consist of neurons that are interconnected using weights[18]. It is essentially a graph that has nodes with weighted edges that connect the nodes. The neurons do a linear transformation based on the incoming weights and an additional bias. In order to also be able to learn non-linear models, the neuron also incorporates what's called an activation function. If we call the input to the neuron x, the weight from the previous neuron w, the bias b and activation function f. The output y of a neuron is calculated as you can see in Figure 6.

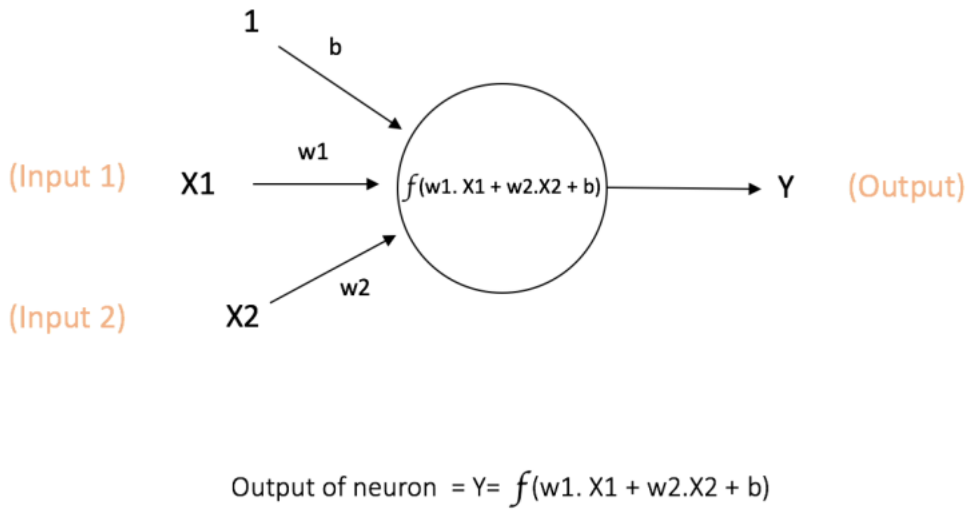


Figure 6: Neuron with multiple input weights

The weight, w is essentially the coefficient, the bias, b is the additive constant of the of the transformation. As you can see in Figure 6, neurons are interconnected with other neurons and typically have a high number of weights connected to it. w and x are therefore often described as vectors.

2.2.2 Layers

Our neural network features a 3-layer architecture. Firstly there's the input layer, this is the layer we we receive our input data. In our case, this are the values of the pixels in the images. The input layer will consist of all the pixels in the image, it will have input dimensions of w x h. If the image is fed into the network in RGB colors, the input layer will have dimensions of w x h x 3.

There's also a layer called the *output* layer. This is where the final predictions are made. This depends heavily on the goal of the network

There's also the so-called *hidden* layer. This is where our network architecture is going to vary down the road. The hidden layer can actually consist of multiple layers. The reason that it's called the hidden layer, is because of the fact that it's not visible to the user of the network, which only interacts with the input and output layers.

2.2.3 Loss Function

The loss function should be a measure of how far a prediction is from the correct value. When training a neural network, the system tries to get as close to the correct solution as possible, in other words, minimize the loss function.

2.2.4 Backpropagation

Backpropagation is the process of using human labeled data and then adjust the weights connecting the neurons for better accuracy. Backward propagation is similar to *forward propagation* which is used when predicting, but goes the other way. We compute a value δ which can be viewed as the *error* for the given neuron. If $x^{(i)}$ is the input, $y^{(i)}$ is the provided correct prediction, l is the layer, j is the number of the node in the layer, z is the weighted sum of inputs for a particular neuron, $cost$ is the cost function that decides how good a certain prediction is, then we define δ the following way:

$$\delta_j^l = \frac{\partial}{\partial z_j^l} cost(i) \tag{2.1}$$

2.2.5 Gradient Descent

δ becomes the amount that we want to change a certain neuron in order to get a better prediction. We can think of it as the gradient for the neural network. When δ has been calculated for all relevant neurons, we can use a gradient descent algorithm to find the minimum for the cost function. Although there are many variations of gradient descent algorithms, they usually try to use the gradient to determine the *steepest* direction that hopefully takes them closer to the global minimum of the loss function [19].

2.2.6 Convolutional Neural Networks

It is possible to send the input image into the input layer and process each pixel individually. Thereby only relations to each pixels individually is learned. When recognizing features in an image it might however be useful to also learn information about how pixels that are spatially close to each other relate. In other words we want to learn further information by looking at pixels that are close to each other.

One common way to do this is to use what is called a convolutional neural network. A convolutional neural network creates a so-called kernel. This is a window of a certain size, typically it's square so it has a size of $N \times N$, where N is odd. The pixels in the image layer then get one weight to a neuron in the next layer for the pixels within it's kernel. If we compare this to a dense layer where every neuron in the input layer is also connected to every neuron in the dense layer, we understand that it doesn't suffer from the curse of dimensionality as much as a dense layer does. The advantage of this is that special information about how pixels close to each other relate is kept while the network still keeps a manageable size.

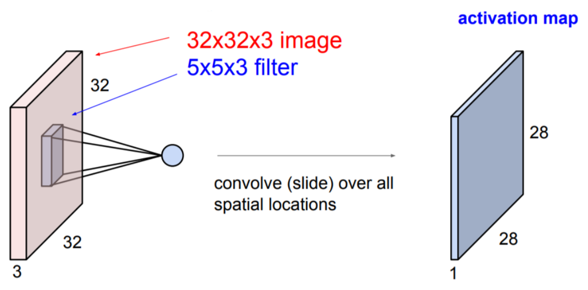


Figure 7: Convolutional layer, image from Stanford Lecture [1]

As you see in Figure 7, the convolutional layer is smaller than the input image. This is due to the fact that a $N \times N$ filter is used, and therefore cannot create corresponding neurons for the edges. The dimension is reduced by $N-1$ neurons. In order to keep the same dimension in the convolutional layer as in the input layer we can introduce a padding rule. We simply say that for all the input pixels that are *outside* the input image, they have the same value as their closest neighbour.

Figure 8 shows a typical convolutional neural network architecture with convolutional-, subsampling- and fully connected layers.

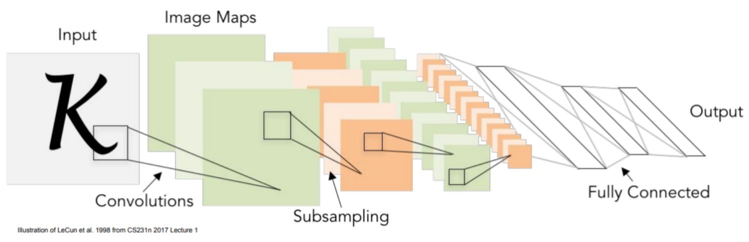

Input
Image Maps
Output
Convolutions
Subsampling
Fully Connected
K
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Figure 8: Convolutional neural network, image from Stanford Lecture [1]

3 Methods and Implementation

3.1 Pattern extraction

3.1.1 Extraction methods

In order to be able to compare patterns accurately some pre processing has to be done beforehand.

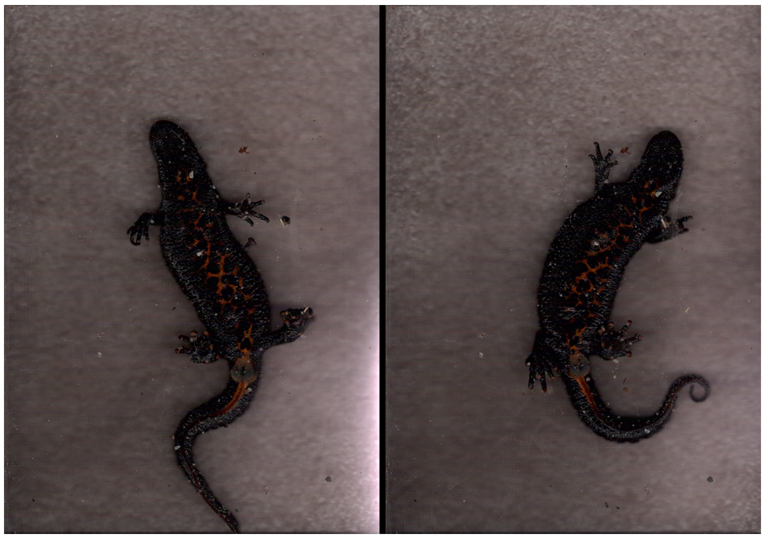


Figure 9: Same salamander posing differently

According to A.J. Ijspeert [20], the salamander can be modeled as having three joints in between the front legs and back legs, as demonstrated in Figure 10. This gets us to a total of 5 joints required to localize the trunk and belly pattern of the salamander. Based on this, we defined the belly pattern to be from the point in between the frontlegs to the point in between the backlegs. We tried to label these 5 points equidistantly along the spine. The the spine will then be estimated using bicubic interpolation [21] of these five points. The five points were defined as you can see in Table 2 and Figure 11. See the blue line in Table 9. We

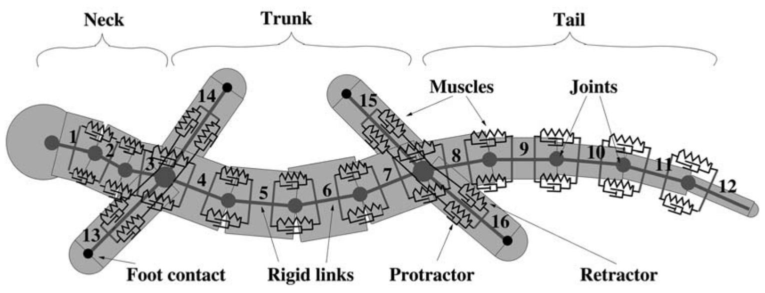


Figure 10: Mechanical model of salamander [20]

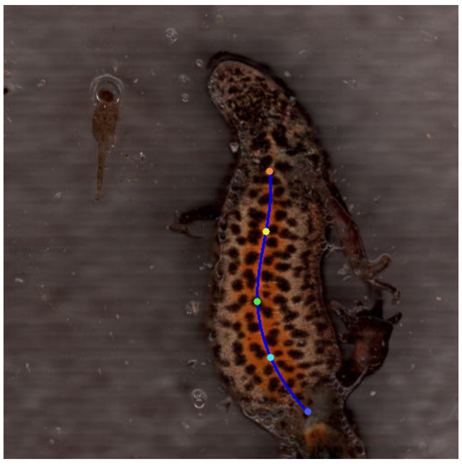


Figure 11: Example of manually labeled salamander

are presenting a number of different methods of varying speed and accuracy to localize and straighten the belly pattern in order to prepare it for matching. In the end we concluded that DeepLabCut [22, 23] (see Section 3.1.6) was the best suited solution in our situation.

Point	Description	Color
Front legs	Point between the front legs	Orange
Belly point 1	First intermediary point along the spine	Yellow
Belly point 2	Second intermediary point along the spine	Green
Belly point 3	Third intermediary point along the spine	Turquoise
Back legs	Point between the back legs	Blue

Table 2: Labeled points on salamander

3.1.2 Manual pose labeling

The simplest method to locate the belly pattern is by manual labeling. A human labeler goes through the images and selects a set of points along the spinal cord. With a large dataset, this can take a large amount of time. Accurate manual labeling took us about 30 seconds, which coincides with the findings of M. Matthé et. al. [3]. When using a dataset with 10000 images, you will have to spend 83.3 hours of manual labeling. Additionally the human labeler is subject to bias and error rates may also increase when working on labeling for a prolonged amount of time.

3.1.3 Segmentation and Skeletonization

This method works by first segmenting out the salamander from the background. And then apply skeletonization, also referred to as a medial axis transfrom to estimate the salamanders spinal cord.

3.1.3.1 Segmentation

The method we initiallly tried was based on having a line that forms a loop surrounding the object you want to create a contour of and then shrink this loop step by step until it forms a contour of the object in the area. This method was found in this article [24].

This method is intended to be used by having user input making the initial line near the object you want to create a contour of. We thought we could use this method by making the line outline the entire image and then find the salamander that way. However this method proved unsuitable for two reasons.

- The main reason was that the algorithm simply took too long since we outlined the whole image, which caused the algorithm to have more points along the line to check and more steps to go through. It usually took around 5-6 minutes to finish running the algorithm with a step and a number of points that was needed in order to identify the salamander. We used twenty points around the salamander this was necessary for the algorithm to get into the nooks around the salamanders extremities. The step size we kept at ten as increasing much beyond that would lead to the algorithm jumping past the thinner parts of the salamander e.g. its tail.

- The other reason was that after the algorithm noise in the background made the contour include much of the background, we tried a couple of methods to workaround this namely blurring and thresholding. Thresholding did not do much as it only highlighted so the algorithm still got stuck. Blurring helped a little as it removed some of the noise while still keeping the salamander distinct enough for the algorithm to stop when the line touched the salamander, and through a lot of fine tuning we believe that this method could have been used to outline the salamander but the time the algorithm in combination with the terrible results we got we looked for other methods.

This method uses a function in openCV called findContours that finds all the contours in the image and puts them into an array. The method also specifies that the images should be binarized for better result so we used Otsu binerization to do so. We then look for the biggest contour that takes up less than 90% of the image, this is because while the salamander is the biggest real contour sometimes the function finds a contour around the edge of the image. This method is further outlined in article [25].

3.1.3.2 Skeletonization

After the salamander is segmented out a medial axis transform is performed [26]. This constructs the skeleton image. After the skeleton image is constructed it is converted to a skeleton network [27]. Additionally, we grouped vertices that are close to each other in proximity together to form the skeleton seen in Figure 12. Getting the belly line then simply consists of extracting the longest edge that is not connected to any leaf vertex.

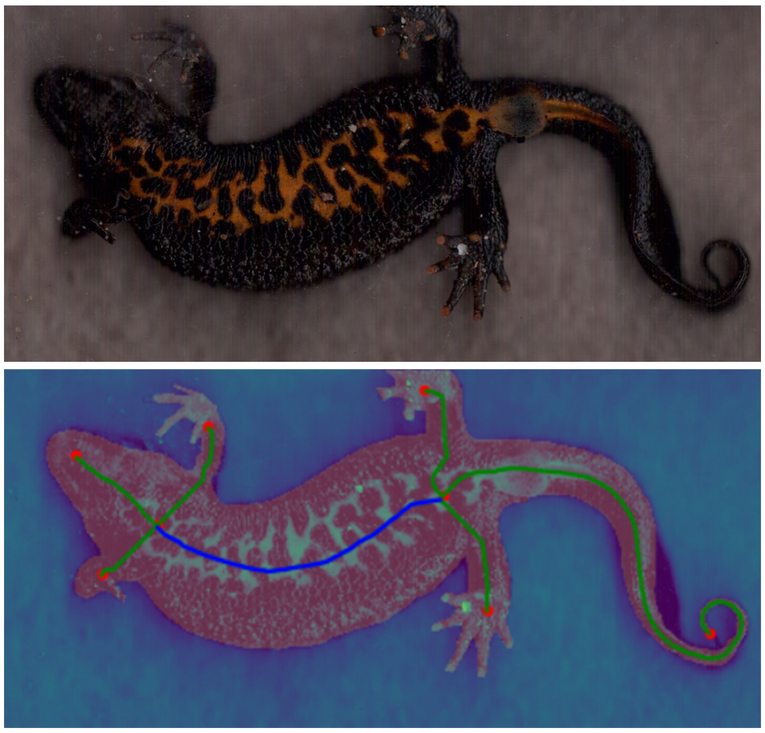


Figure 12: Salamander after manual segmentation and skeletonization

3.1.3.3 Downsides of the method

The Segmentation-Skeletonization method however showed to be quite error-prone due to a number of reasons.

One problem with the Segmentation-Skeletonization method is that it requires the segmentation to be very accurate. If there are small errors in the segmentation, often caused by shadows or other artifacts in the image, this causes the skeletonization to create additional branches as you can see in Figure 13. In this case a part of the tail, marked in blue, is actually erroneously identified as the belly pattern.

If we would be able to attain perfect smooth segmentation as in the example of Figure 12, the same problem also occurs when the salamander does not have it's feet perpendicular to it's body. This causes misalignemnt of the point between the feet which is used as the respective start- and end-point of the belly pattern.

In Figure 12 we see that the salamander is bending and at the same time its's laying down further to one side. This causes the spinal cord and the belly pattern to not align with the medial axis of the segmented salamander. It also causes the pattern to differ depending on how the salamander poses on the image, which in turn negatively affects matching.

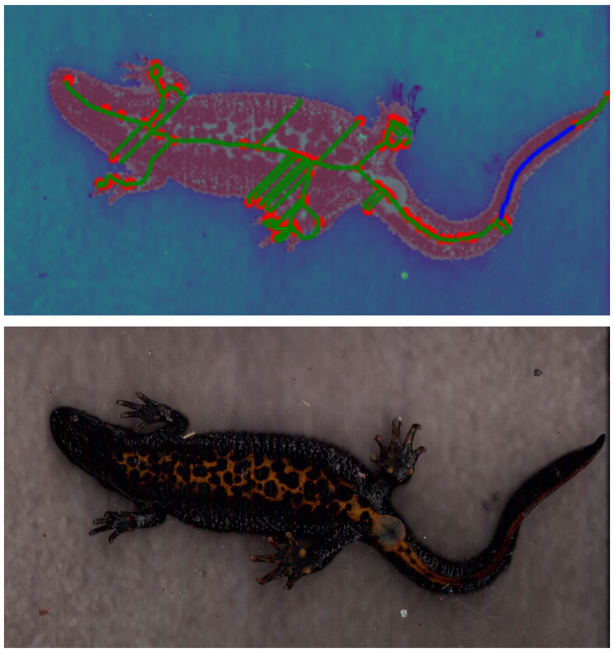


Figure 13: Consequences of minor segmentation errors

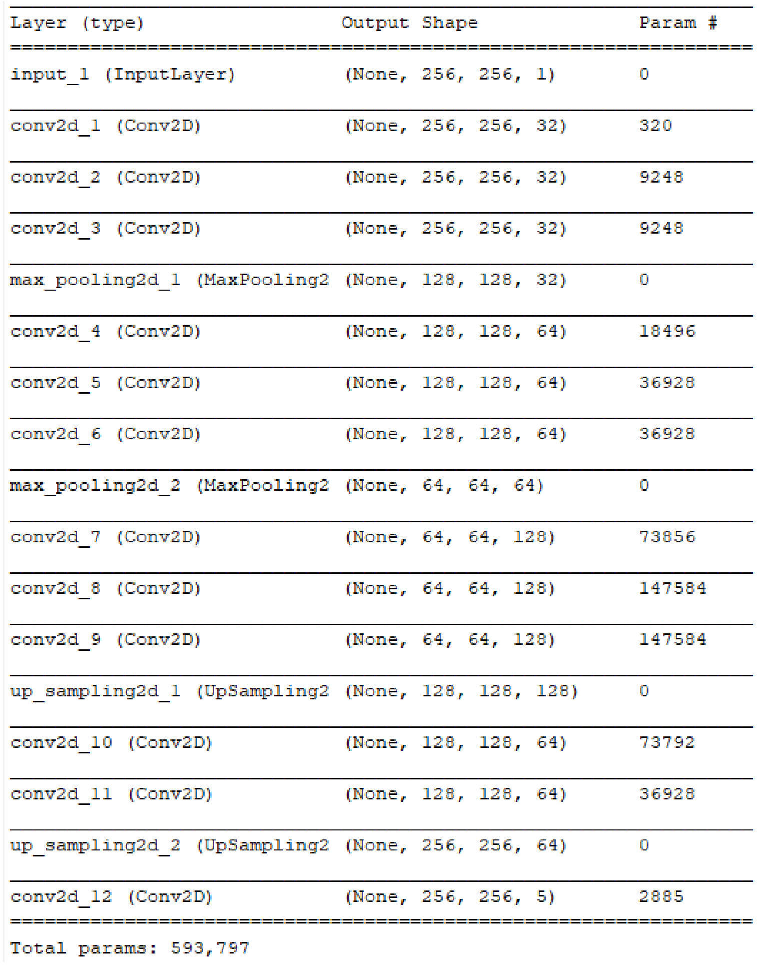
These issues combined led us to look for other methods to automate the localization of the belly pattern.

3.1.4 Data preparation for neural networks

To train the network we extracted 200 smooth newt (Triturus vulgaris) from our dataset (see Section 1.2). As convolutional neural networks perform best when the resolution is as low as possible, we cropped out a 512 x 512 square consisting of the main body of the salamander. We cropped the images by using a laplace transform to get edges and use skin color detection [28] to extract the right part of the image automatically.

3.1.5 Pose estimation using LEAP

In order to more accurately extract the belly pattern. We looked into a system called LEAP [29]. This is a convolutional neural network developed to estimate pose quickly, to be used in realtime applications. As the creators of the network recommended providing small grayscale images of resolution 196 x 196 or 256 x 256, we converted the images to grayscale and re-sized [30] them to 256 x 256 using bicubic interpolation. We split our labeled images into a training set 90% and test set 10% we trained the network with a batch size of 16, and ran 50 batches per epoch, for a total of 15 epochs. We tried augmenting the data using random rotations (+-45°) and mirroring, which is not recommended if the data is not translationally (which it was not in our case). We confirmed that the results were worse with these augmentations as training loss plateaued already in the 2nd epoch. Therefore we did not do any data



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 256, 256, 1)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 32)      320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 256, 256, 32)      9248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 256, 256, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 128, 128, 64)      18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 128, 128, 64)      73792
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 128, 128, 64)      36928
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 256, 256, 64)      0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 256, 256, 5)       2885
=================================================================
Total params: 593,797
```

Figure 14: Network architecture of LEAP

augmentation. We used 10% of the images in each batch for batch-level validation, and used 10 of the 50 batches for epoch-level validation. The training finished in just under 5 minutes using our hardware which is outlined in Table 1.

3.1.5.1 Network architecture

The network architecture, termed leap_cnn by it's creators, is a convolutional neural network architecture, see Section 2.2.6. The relatively simple network architecture of LEAP is demonstrated in the summary shown in Figure 14 which is produced by Keras' summary method [31].

As we can see in Figure 14, LEAP starts with 3 consecutive convolution layers with kernels 3x3 and with a feature dimension of 32. After this a max-pooling layer with a stride of 2 is added, reducing the image dimension to 128 x 128 and. The same process is repeated to reduce the dimension to 64 x 64 and the feature dimension is increased to 128. The image dimensions are then increased to 128 again by a transposed convolution used for upsampling, and a refinement layer in the end. The final layer in the model are the confidence maps produced for each of the five points that we want to predict. The maximum of these confidence maps is the point that is predicted.

All layers use ReLU activation. For it's gradient descent (see Section 2.2.5) the system uses AMSGrad [32] which starts training with ADAM [33] and switches to the stochastic gradient descent after some time. The entire network has a total of 594K trainable parameters.

3.1.5.2 Results

The error is calculated by taking the euclidean distance between the point predicted by LEAP and the human labeled points. The system was evaluated on 10 randomly selected images which you can see in Table 9.

Table 3: LEAP testing results

Metric	Error mean	Standard deviation
Front legs	106.9 px	103.8 px
Belly point 1	68.7 px	45.8 px
Belly point 2	37.2 px	26.3 px
Belly point 3	57.0 px	61.0 px
Back legs	65.4 px	96.4 px
All	**66.0 px**	**72.8 px**

LEAP was able to predict images at a rate of **107.3 FPS**.

From Table 3 we see that error rates are relativelty high, with an average euclidean error of 66.0px which is about the width of a typical salamander in our images. As you can see in Table 9, LEAP was able to generate useful results in only 1 out of our 10 testing images. Note that test image 2 in Table 9 is labeled inversely. It is able to predict the location of the belly pattern on some basic images, but fails quickly if there is rotation and/or bend in the images.

3.1.5.3 Discussion

As these results were not satisfactory for a production-level system, we had two options:

- Tweak the network and label more images to generalize.
- Look for another more flexible network architecture.

As the results of LEAP were as poor as they were, and they claim that "100 frames results in 308 85% of the frames within 2.5 pixel error", we concluded that the neural network was not flexible enough for our data and use case.

There are multiple possible reasons why LEAP was not able to produce satisfactory results.

LEAP was optimized for translationally and rotationally aligned datasets, which means that all salamanders should be in the center of the image and be facing in the same direction. They recorded a video and calculated the direction of movement and were thereby able to extract rotationally and translationally aligned data. As we however only operate on single images, this was no option for us. The fact that LEAP is "not robust to data variance such as rotations" is also been confirmed by other sources [34]. Another reason is that LEAP is designed to make fast predictions to be used in real-time high speed systems which run at 185 Hz. Achieving this speed, LEAP has a relatively small network size which sacrifices flexibility. For these reasons we decided to look for deeper and more adaptable neural networks.

3.1.6 Pose estimation using DeepLabCut

3.1.6.1 Overview

Since the results we achieved with LEAP [29] were not satisfactory, we investigated the usefulness DeepLabCut [22, 23] for our case. DeepLabCut is a tool for markerless pose estimation of user-defined body parts with deep learning. In the abstract of the paper it is claims the following: "even when only a small number of frames are labeled ($\sim$200), the algorithm achieves excellent tracking performance on test frames that is comparable to human accuracy.".

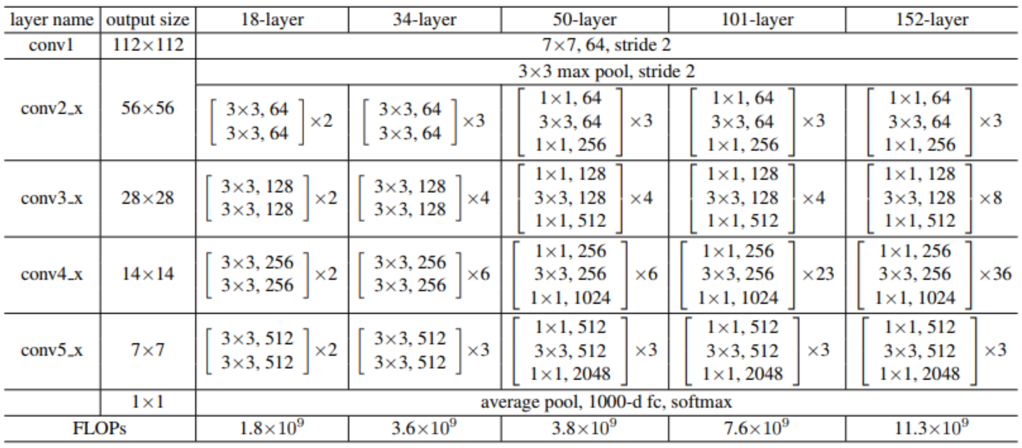
In order to test the accuracy of DeepLabCut, we provided it with the 200 labeled images (see Section 3.1.4). We provided the images in a 512 x 512 RGB format. The labeled images were split into 95% training and 5% testing sets. We did data augmentation by mirroring the images, applying random rotations (+-30°) and resizing (+-25%) the images and labels. Additionally we used pretrained weights for the backbone, which is a deep residual network, ResNet [35], that has run 5000 iterations on the ImageNet [36] dataset, which according to it's authors makes it able to accurately predict pose with only 200 labeled images [22].

It is recommended to train the network for at least 200 000 iterations in order to arrive at a plateau of the loss function [23]. We trained the network for 350 000 iterations which took 12 hours using our hardware (see Table 1).

3.1.6.2 Network architecture

DeepLabCut uses Huber loss as it's loss function (see Section 2.2.3). Huber loss is less sensitive to outlier data than the mean-squared-error loss that is used in LEAP [29], as it is linear instead of quadratic for large errors.

A residual block, as it can be seen in Figure 16, consists of 3 convolutional layers (see Section 2.2.6) with ReLU activation in between them. The arrow going outside the block in Figure 16 indicates the *shortcut*. If the block is not needed the weights can approach zero so the



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 15: ResNet network architectures

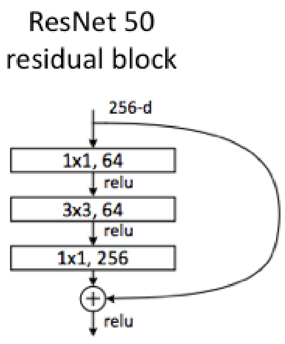


Figure 16: First residual block used by the 50, 101 and 151 layer ResNet models [37]

block uses the shortcut, also called the *identity function*, instead. These shortcuts allow the gradients to propagate through unused blocks easily. This solves the vanishing and exploding gradient problem which are common problems for neural networks of this depth [38, 39]. These residual blocks are then stacked on top of each other, as you can see in Figure 15, in different ways depending on the network size you want. We used the 50-layer model which we found to give a reasonable balance between speed and accuracy.

The ResNet architecture was originally proposed by Microsoft in 2015 [35]. It's 50-layer version, has a a total of 23.5M trainable parameters. Instead of a classification layer at the end of ResNet, DeepLabCut uses "deconvolutional layers to upsample the layers and produce spatial probability densities" [35], similarly to how it is done in LEAP (see Section 3.1.5.1).

3.1.6.3 Results

The error is calculated by taking the euclidean distance between the point predicted by DeepLabCut and the human labeled points. The system was evaluated on 10 randomly selected images which you can see in Table 9.

Table 4: DeepLabCut testing results

Metric	Error mean	Standard deviation
Front legs	6.4 px	2.9 px
Belly point 1	9.3 px	9.8 px
Belly point 2	10.5 px	5.8 px
Belly point 3	8.1 px	4.4 px
Back leg	3.3 px	1.9 px
All	**7.5 px**	**6.2 px**

DeepLabCut was able to predict images at a rate of **5.2 FPS**.

3.1.6.4 Discussion

Although DeepLabCut at 5.2 FPS is considerably slower than LEAP at 107.3 FPS, we also have a better accuracy with a significant margin. From the results we see in Table 4 that we have a error mean of 7.5 px. We also observe that the endpoints have a considerably lower error mean than the three belly points. In Section 3.1.1 we said that we tried to place the 3 points between the front legs and the backlegs equidistantly along the spinal cord. During labeling there was no enforcement that made these points equidistant, it was just by eye. It is therefore reasonable to assume that the labeled points where not perfectly equidistantly aligned although they were relatively accurately placed on the spinal cord. This causes some fluctuations along the spinal cord. This does increase the error mean, but does not affect the resulting bicubic interpolation (see Section 3.1.1) as much. We could also validate this by simple visual inspection by comparing the bicubic interpolation of the manually selected points to the bicubic interpolation of the points selected by the neural network in Table 9.

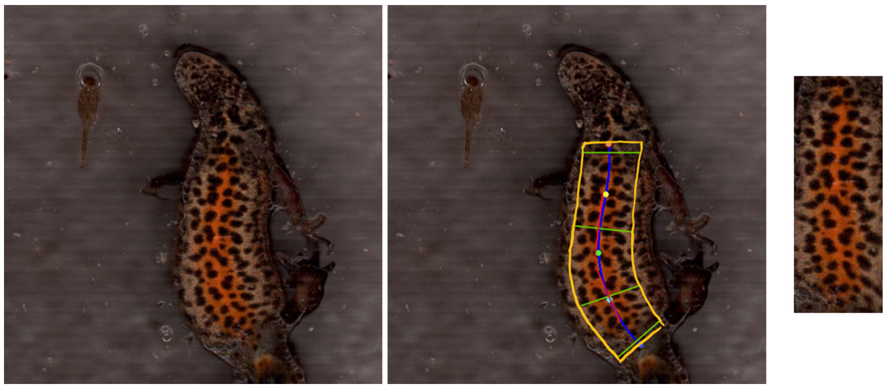


Figure 17: Straightening using DeepLabCut and the derivative of the interpolated spinal curve

After training for 350 000 iterations we had a big difference between our training loss and testing loss which may indicate that the model was overfitting. One way to solve this would be to supply more training data in the form of labeled frames. It could however also be a consequence of human variability in labeling. In order to improve accuracy one could enforce equidistant alignment of the points during labeling. Another thing that should increase accuracy is to replace the ResNet-50 backbone with a ResNet-101 backbone [40]. ResNet-101 features 23 instead of 6 blocks in the conv4_x layer (see Figure 15), providing an even deeper architecture. This would however come at the cost of prediction and training speed.

As the results however were of sufficient accuracy for our application, we decided to focus our time in other parts of the project.

3.1.7 Straightening

As the input image was required to be a straight image, see Section 3.1.1, we had to convert it into a rectangular image, with the spinal cord of the salamander in the center. In order to achieve this, we implemented a nonlinear transformation based on the gradient of the interpolated curve.

The straightening works by using the remap method in OpenCV's image processing module [41, 42]. The remap function works by providing it with 2 maps, one for the x coordinate and one for the y-coordinate. Assuming we're doing a remapping from a source image with dimensions $w_s \times \mathrm{h}_s$ to a destination image with dimensions $w_d \times \mathrm{h}_d$. The maps describe the floating point coordinates of the source image, x_s and y_s, correlating with the integer points x_d and y_d. x_d and y_d are provided to the remap function as two separate $w_d \times \mathrm{h}_d$ maps. As x_s and y_s are floating point numbers, we used bilinear interpolation to estimate their pixel value.

As we can see in Figure 17, the remapping is done by taking the derivative of the curve. In order to form a rectangular image of the curved belly pattern, we're using the derivative $\frac{dy}{dx}$ (red tangent line) as a reference for what is going to be straight for a particular vertical fragment (green lines). The green line will be parallel to the y-axis in the target image while the blue line will be parallel the x-axis. The area within the yellow boundary will effectively be mapped to a straight rectangle as you can see in Figure 17.

3.2 Image processing

For testing the matching algorithms 49 images of the northern crested newt (Triturus cristatus) were extracted from the original dataset 1.2.

3.2.1 Feature based method

There are many different feature extraction method's depending on the objective of the application as described in 2.1.1. In this project we decided to test 4 popular feature based methods, ORB [12] (Oriented FAST and rotated BRIEF), SURF [14] (Speeded up robust features), SIFT [13] (Scale-invariant feature transform) and DSIFT (Dense SIFT). All of these method's are commonly used in computer vision to detect and describe local features in a image.

3.2.1.1 ORB, SIFT and SURF

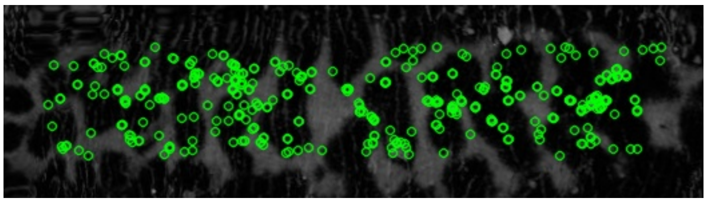


Figure 18: ORB keypoints

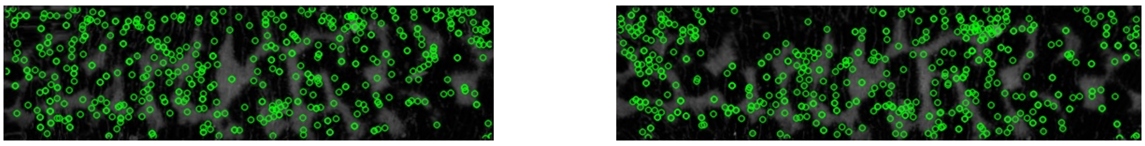


Figure 19: Two images from same individual but different amount of keypoints and locations.

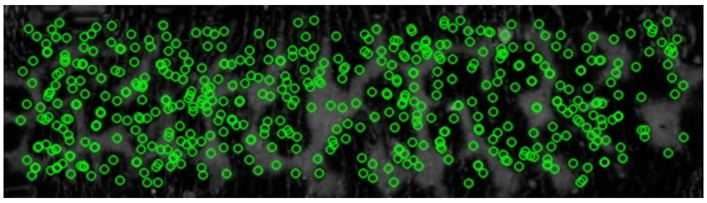


Figure 20: SURF keypoints

SIFT, SURF and ORB calculates keypoints based on a points of interest algorithms such as DoG. The problem with these keypoints in images with repeated patterns is that the keypoints and their descriptors could be very similar to other individuals so the matching algorithm cant tell them apart. Keypoints can have many different locations in the image even if the images is from the same individual as seen in Figure 18 and 20. Figure 19 shows the keypoints calculated from the SIFT algorithm and using these keypoints and their descriptors could cause many false positives by the matching algorithm and therefore not suitable for this type of project.

3.2.1.2 DSIFT

Dense SIFT, is basically the same as SIFT but instead of using DoG (Difference of Gaussian Filtering) to find the keypoints you can specify the keypoints yourself. This method is especially useful when the input image have repeated patterns and regular shapes. We specified how many keypoints we wanted manually with the OpenCV function *cv2.keypoint()* before calculating the descriptors with the function *cv2.xfeatures2d.SIFT_create.compute*. This helped us to have the same amount of keypoints and distance between them in every image as seen in Figure 21.

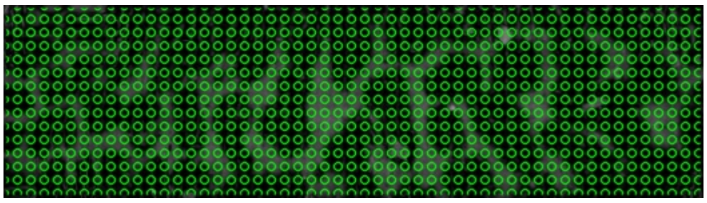


Figure 21: DSIFT keypoints

3.2.1.3 HOG

This technique divides the input image into small square cells and then computes the histogram of gradient directions or edge directions. The gradient directions is the directional change in the intensity or color of the image as seen in 22.
The HOG technique is a fast technique compared to other techniques because of simple computations. The output will be a vector that contains all of the important information about the input image.

Intel describes the implementation of the HOG algorithm as follow [44]:

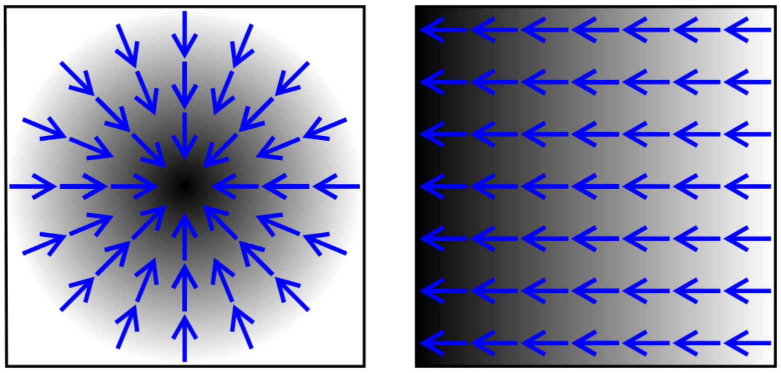


Figure 22: Gradient directions example, image from Wikipedia [43]

1. Divide the image into small connected regions, this is called cells. For each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell.
2. Discretize each cell into angular bins according to the gradient orientation.
3. Each cell's pixel contributes weighted gradient to its corresponding angular bin.
4. Groups of adjacent celled are considered as spatial regions. This is called blocks. The grouping of cells into a block is the basics for grouping and normalization of histograms.
5. Normalized group of histograms represents the block histogram. The set of these block histogram represent the descriptor.

Figure 23 shows the HOG algorithm implementation.

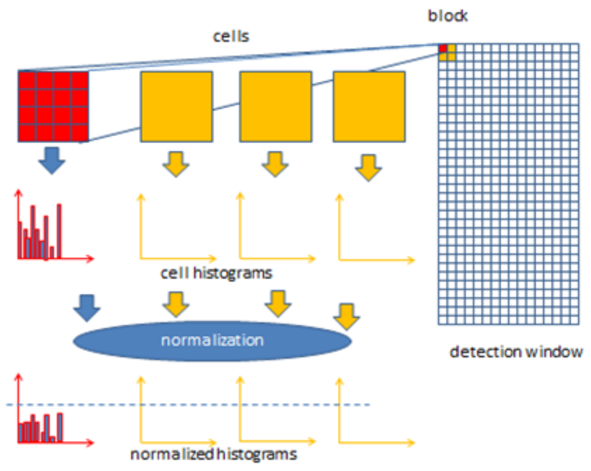


Figure 23: HOG algorithm [44]

When using a HOG function in any programming language the function will return a feature vector 24.

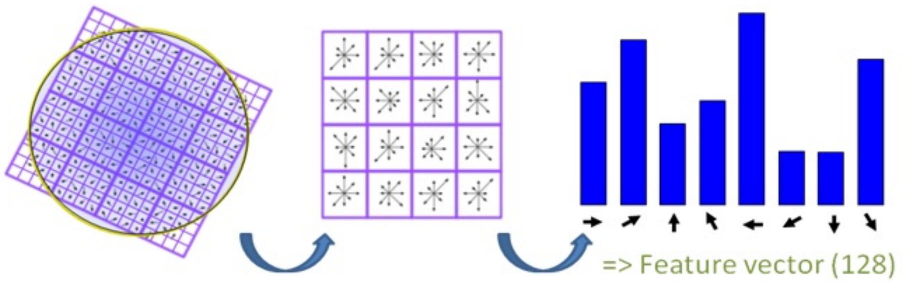


Figure 24: Gradient orientation to vector example [45]

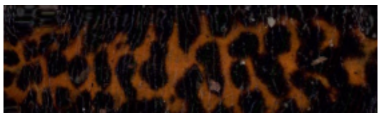
The HOG divides the input into smaller cells and computes a gradient direction histogram with different directions and returns a vector that contains all of this data as shown in Figure 24. This vector will be the descriptor for one keypoint in the input image.

3.2.2 Preprocessing for pixel based method

The pixel method was one of the two main methods of comparing amphibians that we could find, in addition it is the method used by AmphIdent the only one we could find that is commercially available. The method itself revolves around comparing the images pixel by pixel and looking at the difference to make an average pixel difference between the images to find matches. But before one starts comparing the images they go through two main prepossessing steps, namely resizing and thresholding, which we are going to copy.

3.2.2.1 Resizing

In order to use this method of comparison the images need to have the same number of pixels, therefore we resized them to 80x280 using OpenCV's cv2.remap method [41] with bilinear interpolation. In addition to making the images the same size this also shrinks them by roughly 75% which makes the comparison algorithm faster. AmphIdent uses images of size 80x320, however we found that this was a bit too long for our images.


(a) Resize example 1

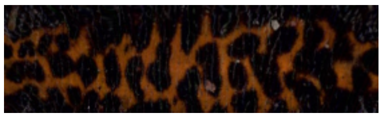

(b) Resize example 2

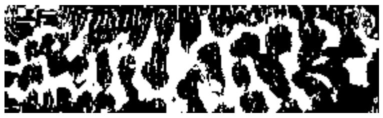
Figure 25: Resize examples

3.2.2.2 Thresholding

One thing we do to increase the spread of the algorithm is binarize the image, this makes it so there is only one colour channel instead of three. We do not lose anything by doing this because the only difference we care about between the images is whether its a spot from the pattern or not. Thresholding is a very simple method for image segmentation. Thresholding takes a grayscale or colour image and creates a binary image as seen in Figure 5.

We do not know what method AmphIdent uses for their binarization so we tried th different methods we outlined in 2.1.1.1.

- Simple thresholding works well, however with the variety if lighting we have in our images it proves hard to find a good value for the threshold that works for all of the images. Images 26a and 26b were made using a threshold value of 120


(a) Simple binarization example 1

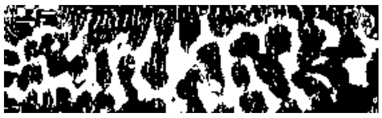
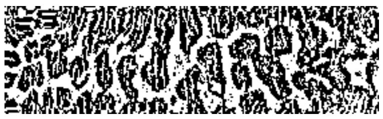

(b) Simple binarization example 2

Figure 26: Simple binerization examples

- Adaptive thresholding was not useful for this purpose as we are trying to highlight bigger patterns which when using this method becomes fragmented as pixels in the pattern that are slightly brighter the surrounding pixels become white as can be sees in figure27.


(a) Adaptive example 1

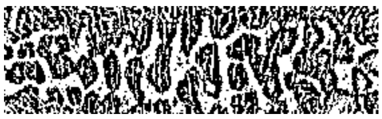
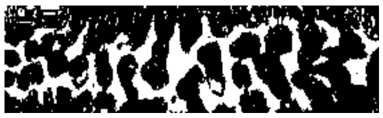

(b) Adaptive example 2

Figure 27: Adaptive examples

- Otsu binarization worked very well across the images we tested as our images are bimodal, which makes it easy for the algorithm to calculate a good threshold value.


(a) Otsu example 1

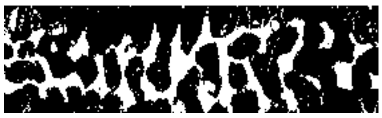

(b) Otsu example 2

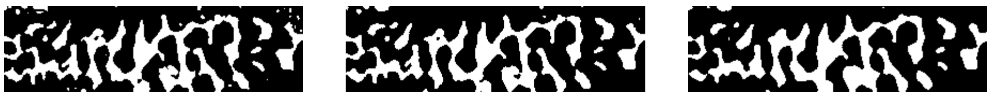
Figure 28: Otsu examples

3.2.2.3 Blurring

There were some inaccuracies in the binarized images caused by noise in the original image. This noise would appear as white or black dots where there should not be any. In order to reduce this we tried to blur the images to smooth out the noise, however there is a real possibility that blurring the images would compromise the edges of the pattern. We therefore tested and compared the methods of blurring from section 2.1.1.2 with the goal of reducing as much noise as possible without compromising the pattern quality. In addition blurring the image would make the peaks in the histogram more distinct which should make the thresholding algorithm yield better results.

- Image 29a is the result of using Gaussian blurring with a kernel size of five.
- Image 29b is the result of using Median blurring also with a kernel size of five.
- Image 29c is the result of using bilateral filtering using sigma colour of twenty and sigma space of seventy five.

While all of these methods helped to some degree we would say that the median blur removed the most noise with the least effect on the detail of the pattern. However even this method resulted in minimal gain and given the evident loss of detail seen in figure 29, it might not be worth using.



(a) Gaussian blur (b) Median blur (c) Bilateral filtering

Figure 29: Binarized blurred images.

3.3 Matching

3.3.1 Brute force matching

This is a very simple but fast matching method. The function *cv2.BFMatcher()* from OpenCV [46] takes two set of data, in this case two descriptors. The function will use the first descriptor as a "template" and then try to find the closest one in the second set. It will then return two objects, the template and the closest one. In these two objects there's something called distance and this will tell us the distance between the two matched points. If the distance are within a certain number (in our case 0.75, this number is commonly used for ratio tests) its a match. In this project we have set a minimum number of matches before we can be sure its a match. This is because of the images are very similar so its possible to get a match even though the two descriptors are from two different animals. We tested many different numbers and with some bad images we will get a low amount of good matches. Since the lowest number in our test set was 15 we have chosen to use this as a minimum. This means the two descriptors needs at least 15 matches before we can be sure the pattern belongs to the same individual animal. Our results using feature descriptors can be found in Table 5 and 10.

3.3.2 Pixel based matching

The method for matching in the pixel based method is really simple. You calculate the average pixel difference adding up the differences of all the pixels and dividing by the dimensions of the image. This gives a value between 1 and 255 as these are the two values in our binary images by dividing this value by 255 we get the percentage difference between the two images based on pixels.

One issue with this method is that if the pattern is shifted a little in one of the images which can drastically increase the difference between the images. To solve this AmphIdent translates the image little by little in each direction and then runs the comparison algorithm over and over again and picks the best score. We used the same approach and shifted our image 20 pixels left and right with a step size of 2 and up and down 10 pixels and picking the best result.

In order to avoid finding false matches we had to set a threshold for how much similarity would be considered a match, the number we ended up with was 75% as when testing the algorithm 72.7 was the lowest result that was given of a true match. In other words the algorithm finds the best match above 70 percent to find a match.

	Correct result	Match not found	False positive	Potential false positives
DSIFT	45	4	0	0
%	91.84%	8.16%	0.00%	
Pixel	21	9	19	511
%	42.86%	18.37%	38.78%	

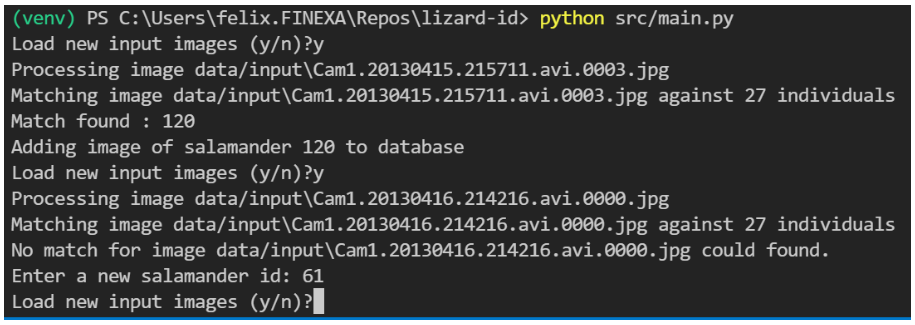
Table 5: Match rates on DSIFT and pixel based method

With the good results using descriptors from DSIFT and brute force matcher compared to the pixel method as seen in Figure 5, we decided to use DSIFT and brute force matching in the final implementation.

4 Implementation and results

4.1 System implementation

To demonstrate the usability of these methods, we wrote a command line interface application i Python. The program reads images from an input folder, trying to match them by all individuals that are inside it's database. The database is saved and loaded from a JSON file. When provided with images, it either tells you the ID of the salamander you tried to match and adds the image to the salamander in the database. If no match is found, the user is asked to provide the system with a unique ID. A new salamander is then created and the image is imported into the database.



```
(venv) PS C:\Users\felix.FINEXA\Repos\lizard-id> python src/main.py
Load new input images (y/n)?y
Processing image data/input\Cam1.20130415.215711.avi.0003.jpg
Matching image data/input\Cam1.20130415.215711.avi.0003.jpg against 27 individuals
Match found : 120
Adding image of salamander 120 to database
Load new input images (y/n)?y
Processing image data/input\Cam1.20130416.214216.avi.0000.jpg
Matching image data/input\Cam1.20130416.214216.avi.0000.jpg against 27 individuals
No match for image data/input\Cam1.20130416.214216.avi.0000.jpg could found.
Enter a new salamander id: 61
Load new input images (y/n)?
```

Figure 30: Our software in action

In Figure 30 the command line interface is used to first match a salamander that is already in the database. Secondly a new salamander, that has not been seen by the system before, is provided.

4.2 Results

In Table 6 we summarize the total speed of all preprocessing steps for a single image for the system as a whole. From a raw image provided until it is inserted into the database ready for matching. This was evaluated on a GTX 1060 6GB GPU, using a more powerful GPU may increase performance. Additionally these steps can be run in parallel by running them on different cores and GPUs, yielding 4x a speed increase.

Table 6: Speed of all pre-matching steps in our system

Process	Time required
Automated Cropping (smartcrop.js)	20 ms
Belly localization (DeepLabCut)	192 ms
Straightening	250ms
Extract feature descriptors	45 ms
Total	**507 ms**

Table 7: Time required for preprocessing

# of images	5 000 images	10 000 images	20 000 images
Manual (existing software)	41.6 hours	83.3 hours	166.6 hours
Automated (1 GPU)	42 minutes	1.4 hours	2.8 hours
Automated (4 GPUs)	10.5 minutes	21 minutes	42 minutes

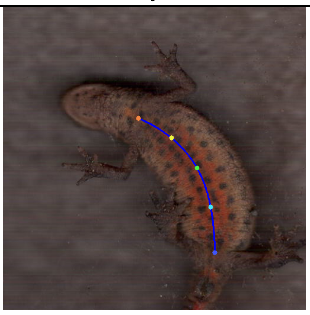
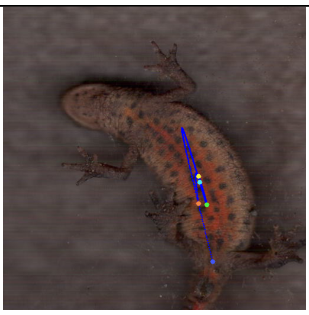
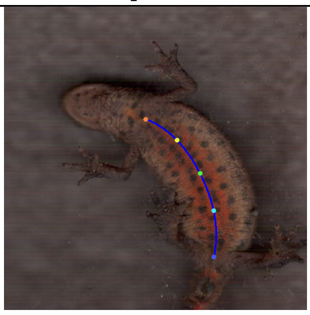
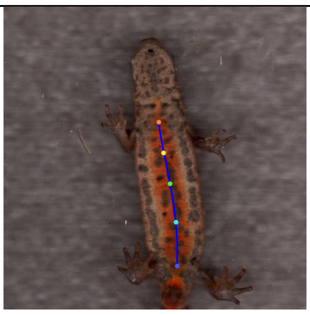
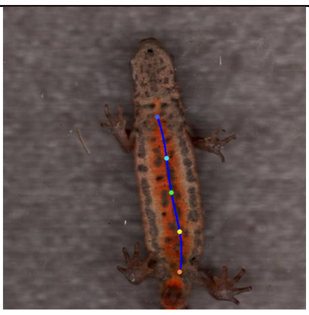
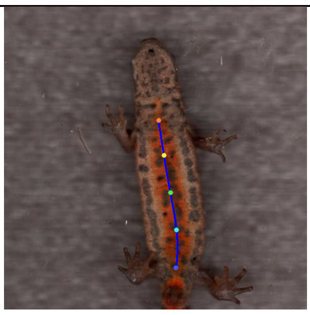
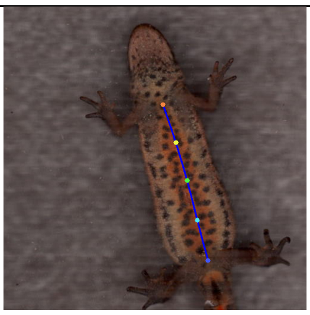
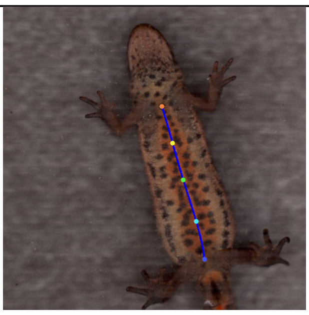
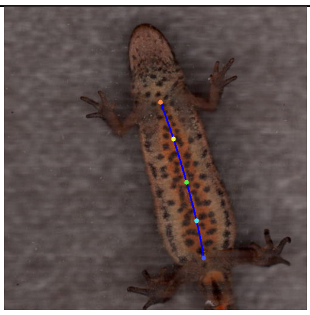
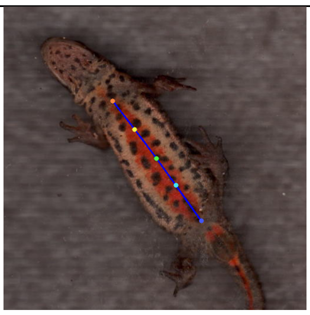
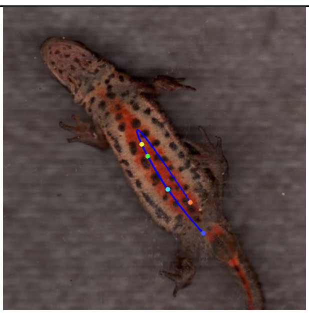
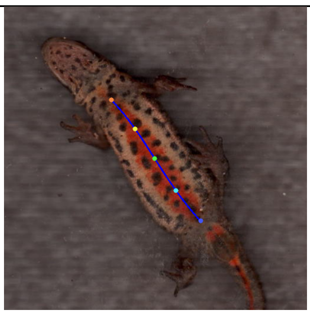
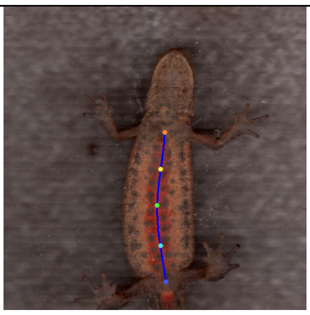
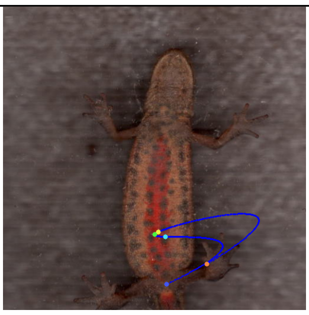
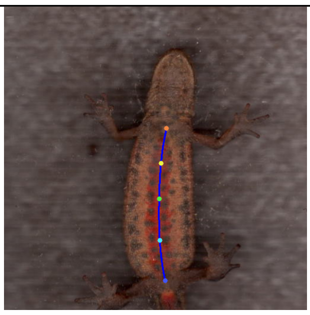
Table 7 shows the time required to preprocess images before the matching can begin. We're assuming that it takes 30 seconds to do manual straightening, which is a metric also used by M. Matthé et al. [3].

Software	Matching	Straightening	Licensing
Aphis	Pixel-based method and SURF[14]	No	Open Source
AmphIdent	Pixel-based method	Manual	Commercial
Wild-ID	SIFT [13]	No	Open Source
I3S Pattern	SURF [14]	Manual	Open Source
Our system	**DSIFT**	**Automatic**	**Open Source**

Table 8: Comparison between existing software and our software

4.2.1 Belly localization

Table 9: Comparison of belly localization methods

#	Manually labeled	LEAP	DeepLabCut
1			
2			
3			
4			
5			

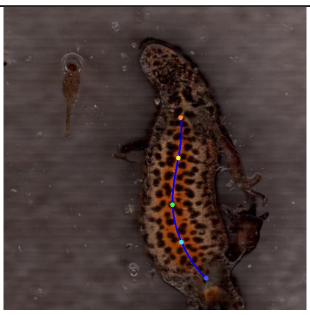
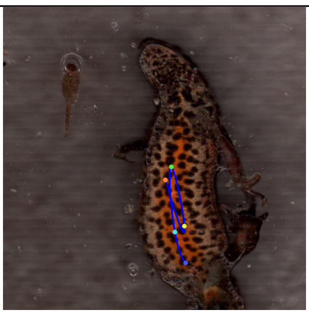
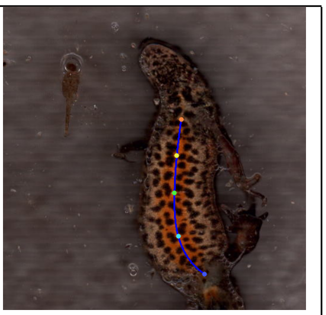
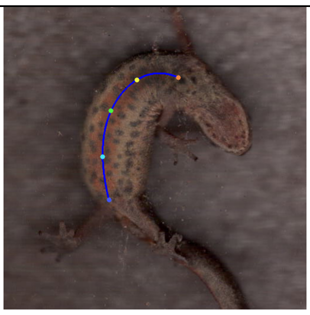
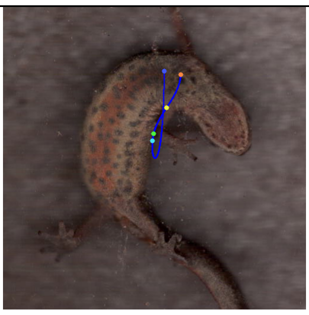
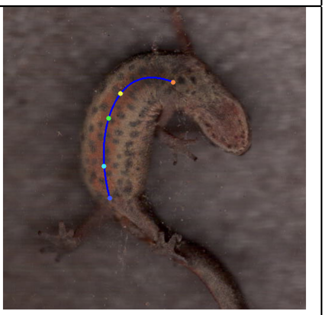
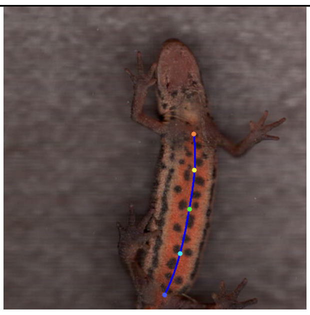
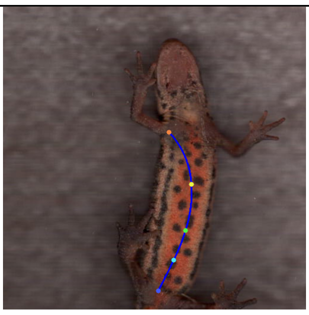
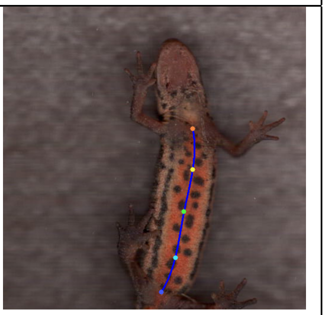
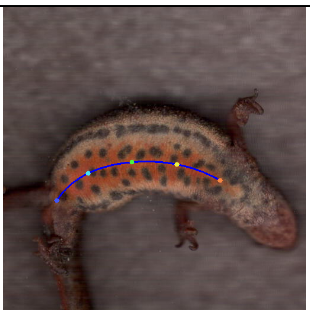
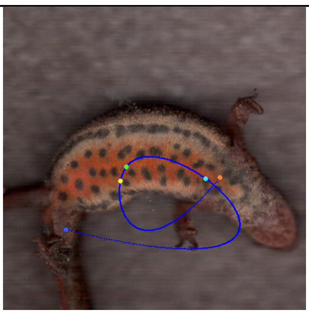
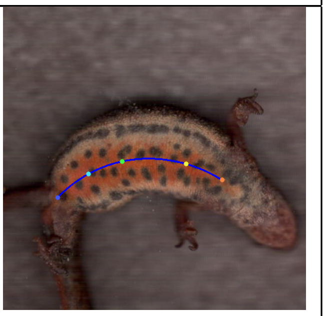
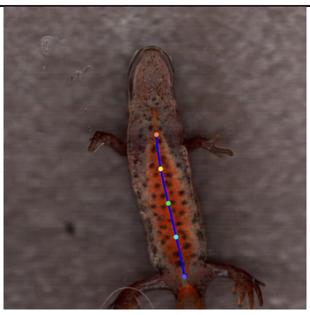
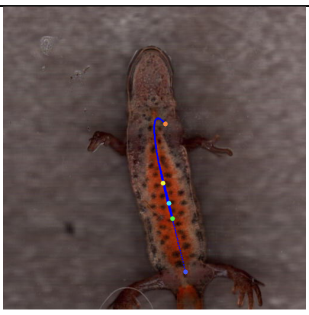
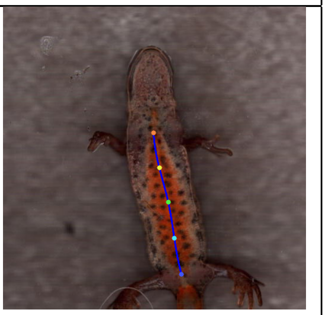
6			
7			
8			
9			
10			

Table 9 compares manually labeled images with the labeling from LEAP and DeepLabCut. These images are from the test set and have not been seen by the neural networks before.

While DeepLabCut is able to give a good estimation the spine on every image, LEAP is only able to label one image correctly.

4.2.2 Matching results on test data

ID	Result using Features	Potential bad matches	Result using pixel	Potential bad matches
120	Found	0	Found	0
120	Found	0	Not found	0
120	Found	0	Found	0
150	Found	0	Found	7
150	Found	0	Found	8
160	Found	0	Found	9
160	Found	0	Found	15
238	Found	0	Bad match	2
238	Found	0	Bad match	9
238	Found	0	Bad match	0
244	Found	0	Bad match	26
244	Found	0	Found	26
245	Found	0	Found	23
245	Found	0	Found	25
246	Found	0	Found	17
246	Found	0	Found	17
247	Found	0	Bad match	27
247	Found	0	Bad match	29
248	Found	0	Bad match	13
248	Found	0	Bad match	9
249	Found	0	Found	12
249	Found	0	Found	15
251	Found	0	Not found	22
251	Found	0	Not found	0
252	Found	0	Bad match	23
252	Found	0	Bad match	17
253	No match to find	0	Bad match	11
255	Not found	0	Bad match	19
255	Not found	0	Bad match	27
256	Not found	0	Bad match	28
256	Not found	0	Bad match	17
260	Found	0	Bad match	13
260	Found	0	Bad match	9
268	Found	0	Not found	0
268	Found	0	Not found	0
271	No match to find	0	No match to find	0
274	No match to find	0	No match to find	0
280	No match to find	0	No match to find	0
285	No match to find	0	No match to find	0
287	No match to find	0	No match to find	0
293	No match to find	0	No match to find	0
294	Found	0	Not found	8
294	Found	0	Not found	17
295	Found	0	Found	1
295	Found	0	Found	1
63	Found	0	Not found	0
63	Found	0	Not found	0
64	Found	0	Bad match	5
64	Found	0	Bad match	4

Table 10: Matching results

Table 10 aggregates the result from running pixel-based and feature-based matching algorithms. Each line is the results from one image it includes the id of the salamander in the image, the result of the algorithm and how many potential bad matches the algorithm found for the image.

- Found means that the algorithm found the correct match.
- No match to find means that there is no match in the system and the algorithm found no match which is correct
- Not found means that the algorithm did not find any match even though one or more exists in the system
- Bad match means that the algorithm found a match of an image of a different salamander.

4.3 Failed cases

4.3.1 Belly localization

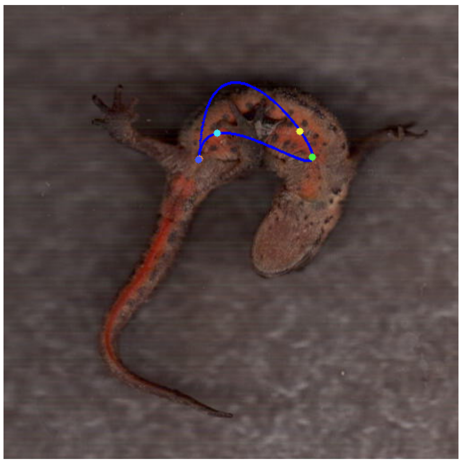


Figure 31: Example of failed pose estimation by DeepLabCut

Figure 31 shows a salamander where the belly pattern is occluded by the salamanders feet, and the pose estimation by DeepLabCut is incorrect.

4.3.2 Matching

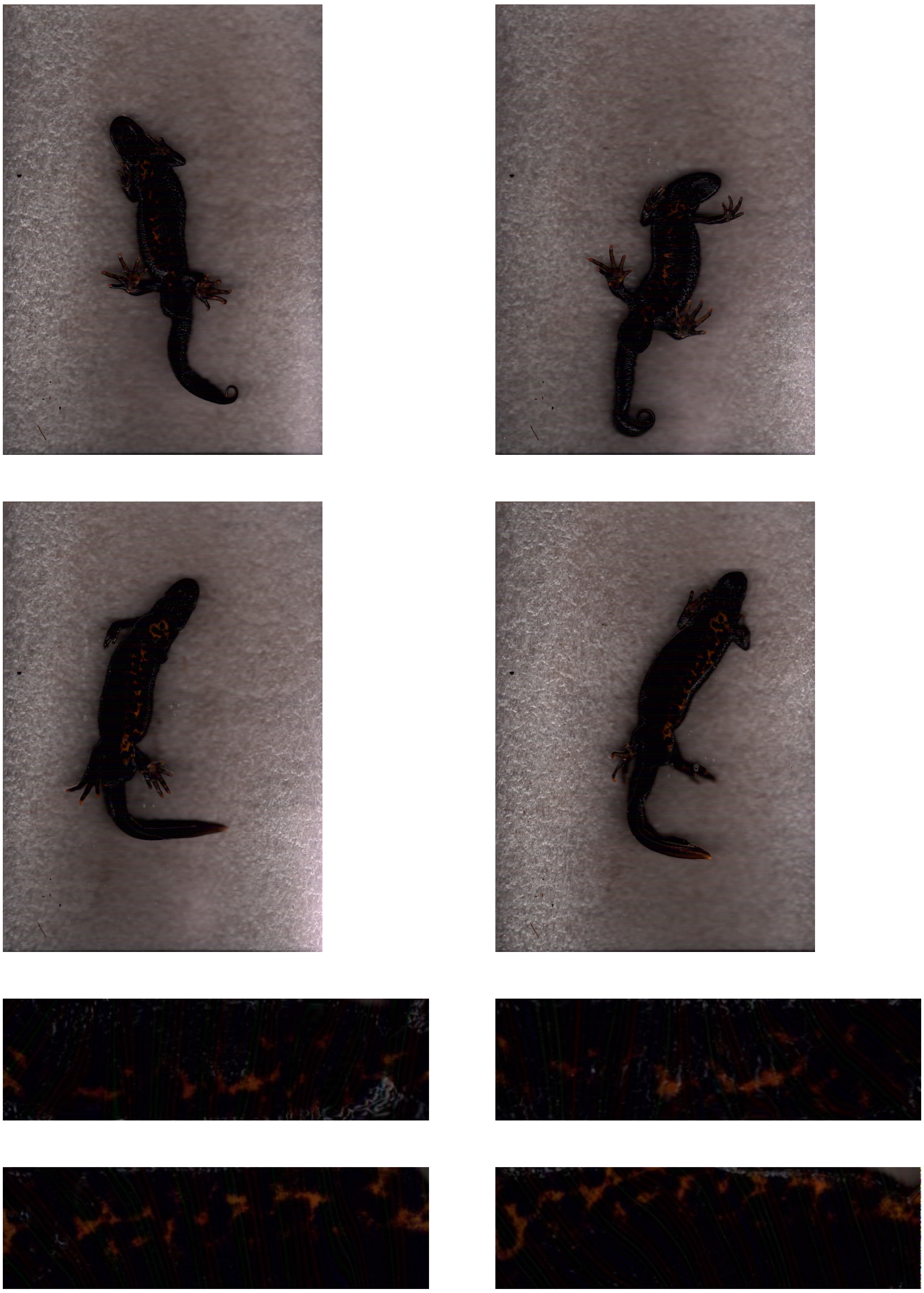


Figure 32: Example images that's too dark for matching

Figure 32 shows images that's too dark. With our implementation of the brute force matcher these images wont get a match because they're too dark. We did not have access to the RAW files during this project to experiment with the brightness etc. to see if that would help getting a match on these images.

5 Discussion

5.1 Belly localization

From the results in Table 3, Table 4 and Table 9 we get an indication that DeepLabCut is far superior to LEAP. This is probably because of the considerably larger neural network that DeepLabCut. While LEAP has 594K trainable parameters, DeepLabCut has a total of 23.5M trainable parameters. That is a 40 fold increase in number of parametrs. This is reflected in the speed with LEAP at 107 FPS while DeepLabCut can only do 5.2 FPS. The accuracy however also mirrors this with respectively 66.0 px error for LEAP and 7.5 pixels error for DeepLabCut. Some of the error may also come from human variability in the labeled data as the spots are not always exactly markable.

Although DeepLabCut reports relatively low error we observe in Figure 31 that it cannot correctly estimate pose on all images. When the neural network is provided with very confusing images, like occlusion of the belly pattern by the feet of the salamander, it is not able to estimate pose correctly. This can however be solved by training the neural networks on these tricky images specifically so it can also learn these cases. In this specific case we will not be able to use the belly pattern for reliable matching as big parts of the belly pattern are occluded.

5.2 Matching

5.2.1 Pixel by pixel method

This method showed a lot of early promise as the first test was to find matches for ten images from a pool of another ten images consisting of of pairs of the original ten. When we moved to a bigger pool trying to find the same 10 matches in a pool of 48 images the best match of each image was still a real match, however when we looked at all the the match numbers there were plenty of images that gave potential matches meaning that if there had been no match in the system it would have given a false match as a result. These false positives varied in severity and we consider increasing the threshold in order to filter more of them out. However some of the false positives reached as high as 92%, and moving the threshold that high would result in most of the true matches as not good enough.

5.2.2 Brute force matching

Our results with brute force matching have been higher than expected. As shown in Table 10 we have achieved a high accuracy. In some of our test data we only have one image of an individual and this is marked as No match to find in the table. Some of the images that we got from NINA had very bad quality in terms of image brightness and overall quality as seen in Figure 32 and were almost impossible to identify. With our method it is possible to identify these images but because of the minimum limit of good matches we decided not to let these images go through. This is because if we lower the limit its a higher chance that we get a false positive match on the other images. We decided that its better to have a higher limit so we're completely sure its a match and rather increase our requirements (brightness and one coloured background) for the images .

In the end of our project NINA have been out in the field taking new images. According to NINA these images will have a much better quality than the images we have used in our project and hopefully match our updated image requirements will increase our matching accuracy.

6 Conclusion

6.1 Summary of the project

As outlined in the Section 1.1, the goal of the project was to propose a solution which:

- Identifies salamanders based on pattern recognition with a high accuracy comparable to existing software.
- Reduces the amount of manual work required to match salamanders by automating the workflow.

To begin with we investigated a range existing softwares that are available, commercially and open source, and outlined their differences and individual advantages and disadvantages. We also got an overview of the state of the art methods and techniques used to solve the problem of matching salamander patterns. Based on the background study conducted, we formulated a problem definition which we were going to solve. Furthermore, we proposed a range of different solutions to the formulated problems with their own advantages and disadvantages. These methods were then evaluated to mitigate to obtain an optimal solution with regard to the project goals. Based on the evaluation the set of methods which best served the project goals, were selected for implementation. The methods selected were the neural network DeepLabCut [22] for pose estimation and Dense SIFT [13] for pattern recognition. Finally, we implemented a software solution using the selected methods.

One contribution of our system is that it uses a matching methods that has a high accuracy on our dataset specific. **The main contribution of our system however, contrary to existing software, is that the system provides a fully automated workflow.** To the best of our knowledge, nobody has proposed this kind of automated solution for salamander recognition before. Recent advancements in deep neural networks allowed us to create this automated solution. Although there is room for improvement, and some cases of failure, we showed evidence that the automated recognition system works on the majority of cases. These methods have been bundled together in a software application that will hopefully serve NINA, and potentially others, in their ongoing research regarding the salamanders.

6.2 Future Work

New features ideas:

Create a GUI (Graphical user interface) and optimize the system for NINA's new RAW images.

Add another method to find and match the descriptors such as SLAM based methods [?] which simultaneously localizes and maps the salamander in real time to create a more accurate and faster matching, even though our solution is performing pretty well.

Also it is possible to add a classification step in the beginning of the pipeline. This would make it possible to classify salamanders into species and gender. This information would then no longer need to be manually provided but can be automatically saved to the database. Additionally it would decrease matching time and increase accuracy as it only has to match with salamanders of the same species and gender. This could be implemented using ResNet [35] which we already use for pose estimation, but was originally built for image classification.

As our proposed solution is a fully automated system, it is also possible to record a video feed using a camera that is installed in a salamander habitat to detect and recognize salamanders in real time.

Another possibility would be to convert the system into a REST Api in order to be able to predict photos from your mobile phone. This would give NINA instant feedback on whether the salamander has been catched before.

This software should work with other species as well, if we insert images of other species the program should be able to extract the patterns of other species. The matching should also be able to handle other species because of the way DSIFT works.

Bibliography

[1] Stanford university: Cs2315 - lecture 6: Training networks part 1. Available at http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture06.pdf, Accessed 17:30, 18 May 2019.

[2] Gibbons, W. J. & Andrews, K. M. 2004. Pit tagging: simple technology at its best. *Bioscience*, 54(5), 447–454.

[3] Matthé, M., Sannolo, M., Winiarski, K., Sluijs, A. S. V. D., Goedbloed, D., Steinfartz, S., & Stachow, U. 2017. Comparison of photo-matching algorithms commonly used for photographic capture-recapture studies. *Ecology and Evolution*, 7(15), 5861–5872. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5552938/, doi:10.1002/ece3.3140.

[4] Management, R. URL: http://www.reijns.com/i3s/download/I3S_download.html.

[5] Bolger, D. T. URL: https://home.dartmouth.edu/faculty-directory/douglas-thomas-bolger.

[6] Amphident. URL: http://www.amphident.de/en/pages/download.html.

[7] d'Estudis Avançats, I. M. URL: http://imedea.uib-csic.es/bc/ecopob/index.php?option=com_content&view=article&id=73&Itemid=89&lang=en.

[8] Ribeiro, S. 04 2014. Using simplecv for seed metadata extraction into xml document. *Iberoamerican Journal of Applied Computing*, 4, 29.

[9] Gonzalez, R. C. & Woods, R. E. 2006. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[10] Edgedetection. Available at https://en.wikipedia.org/wiki/Edge_detection, Accessed 18:04, 19 May 2019.

[11] Corner detection. Available at https://en.wikipedia.org/wiki/Corner_detection, Accessed 18:05, 19 May 2019.

[12] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. R. 2011. Orb: An efficient alternative to sift or surf. In *ICCV*, volume 11, 2. Citeseer.

[13] Lowe, D. G. et al. 1999. Object recognition from local scale-invariant features. In *iccv*, volume 99, 1150–1157.

[14] Bay, H., Tuytelaars, T., & Van Gool, L. 2006. Surf: Speeded up robust features. In *European conference on computer vision*, 404–417. Springer.

[15] Unknown. Thresholding lecture. Available at http://cs.haifa.ac.il/hagit/courses/ip/Lectures/Ip12_Segmentation.pdf.

[16] Blur. Available at https://en.wikipedia.org/wiki/Gaussian_blur, Accessed 18:06, 19 May 2019.

[17] Hough transform. Available at https://en.wikipedia.org/wiki/Hough_transform, Accessed 18:07, 19 May 2019.

[18] Fausett, L., ed. 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[19] Rojas, P. D. R., ed. 1996. *Neural Networks: A Systematic Introduction*. Springer-Verlag.

[20] Ijspeert, A. 06 2001. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological cybernetics*, 84, 331–48. doi:10.1007/s004220000211.

[21] scipy.interpolate.interp1d.

[22] Mathis, A., Mamidanna, P., Abe, T., Cury, K. M., Murthy, V. N., Mathis, M. W., & Bethge, M. 2018. Markerless tracking of user-defined features with deep learning. *arXiv preprint arXiv:1804.03142*.

[23] Nath, T., Mathis, A., Chen, A. C., Patel, A., Bethge, M., & Mathis, M. W. 2018. Using deeplabcut for 3d markerless pose estimation across species and behaviors. *bioRxiv*. URL: https://www.biorxiv.org/content/early/2018/11/24/476531, arXiv:https://www.biorxiv.org/content/early/2018/11/24/476531.full.pdf, doi:10.1101/476531.

[24] Pandey, P. Feb 2019. Image segmentation using python's scikit-image module. URL: https://towardsdatascience.com/image-segmentation-using-pythons-scikit-image-module-533a61ecc980.

[25] Ghoneim, S. Apr 2019. Object detection via color-based image segmentation using python. URL: https://towardsdatascience.com/object-detection-via-color-based-image-segmentation-using-python-e9b7c72f0e11.

[26] Scikit image documentation morphology.skeletonize. Available at https://scikit-image.org/docs/0.15.x/api/skimage.morphology.html#skimage.morphology.skeletonize, Accessed 08:58, 12 May 2019.

[27] Build network from skeleton image (2d-3d). Available at https://github.com/yxdragon/sknw, Accessed 09:05, 12 May 2019.

[28] Content aware image cropping. Available at https://github.com/jwagner/smartcrop.js/, Accessed 10:36 12 May 2019.

[29] Pereira, T., Aldarondo, D., Willmore, L., Kislin, M., Wang, S., Murthy, M., & Shaevitz, J. W. 2018. Fast animal pose estimation using deep neural networks. *bioRxiv*. URL: https://www.biorxiv.org/content/early/2018/05/25/331181, arXiv:https://www.biorxiv.org/content/early/2018/05/25/331181.full.pdf, doi:10.1101/331181.

[30] Opencv documentation - cv2.resize.

[31] Keras - deep learning for humans. URL: https://github.com/keras-team/keras.

[32] Wang, Yijun, Zhou, Pengyu, & Zhong, Wenya. 2018. An optimization strategy based on hybrid algorithm of adam and sgd. *MATEC Web Conf.*, 232, 03007. URL: https://doi.org/10.1051/matecconf/201823203007, doi:10.1051/matecconf/201823203007.

[33] Kingma, D. P. & Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[34] Graving, J. M., Chae, D., Naik, H., Li, L., Koger, B., Costelloe, B. R., & Couzin, I. D. 2019. Fast and robust animal pose estimation. *bioRxiv*. URL: https://www.biorxiv.org/content/early/2019/04/26/620245, arXiv:https://www.biorxiv.org/content/early/2019/04/26/620245.full.pdf, doi:10.1101/620245.

[35] He, K., Zhang, X., Ren, S., & Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

[36] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

[37] First residual block used by the 50, 101 and 151 layer resnet models. Available at https://www.jeremyjordan.me/convnet-architectures/, Accessed 10:52, 15 May 2019.

[38] Hochreiter, S. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02), 107–116.

[39] Pascanu, R., Mikolov, T., & Bengio, Y. 2012. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2.

[40] Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M., & Schiele, B. 2016. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision*, 34–50. Springer.

[41] Opencv: Geometric image transformations - cv2.remap. Accessed 11:13 8 May 2019. URL: https://docs.opencv.org/3.4.2/da/d54/group__imgproc__transform.html#gab75ef31ce5cdfb5c44b6da5f3b908ea4.

[42] Opencv: Remapping. Accessed 11:10 8 May 2019. URL: https://docs.opencv.org/3.4.2/d1/da0/tutorial_remap.html.

[43] Image gradient. Available at https://en.wikipedia.org/wiki/Image_gradient, Accessed 16:00, 15 May 2019.

[44] Hog algorithm. Available at https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor, Accessed 15:02, 13 May 2019.

[45] Hog vector. Available at https://cbutkus.github.io/CS766/, Accessed 15:12, 13 May 2019.

[46] Opencv bfmatcher. Available at https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html, Accessed 16:19, 15 May 2019.

A Project Agreement

NTNU
Norges teknisk-naturvitenskapelige universitet

Vår dato | Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

NORSK INSTITUTT FOR NATURFORSKNING
-NINA (oppdragsgiver), og

JØRGEN BAKKLØKKEN, FELIX SCHOELER,
HUGO NØRHOLM

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 18/1-19 til 16/5-19 .

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): SONY GEORGE

Oppdragsgivers kontaktperson (navn): BØRRE OERVO

Student(er) (signatur): Felix Schaller dato 30.01.19

Hugo Nørholm dato 31/1-19

Jörgen Buklökken dato 31/1-19

dato

Oppdragsgiver (signatur): [illegible] dato 18/1-19

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): dato

B Projectplan

NTNU

Bachelor thesis

BIDAT39

Project plan

Author:
Jørgen Bakløkken
Hugo Nørholm
Felix Schoeler

Supervisor:
Sony George
Co-supervisor:
Marius Pedersen

May 9, 2019

NTNU
Norwegian University of
Science and Technology

Contents

1 Goals and frames

1.1 Background

One of the fields of research at NINA (Norwegian Institute for Nature Research) is salamanders. Salamanders are a group of amphibian and are predators that hunts insects and earthworms. In Norway there are two different species of salamander: northern crested newt and smooth newt. And since they both are a red listed specie its important to increase their population.

NINA are trying to learn more about their life cycle, living area and population. And to get more knowledge they started a pit tagging project in 2010. This means that they have to catch the salamanders in a gently way, inject a pit tag in the salamander and document this into their database. This is very time consuming for NINA and also stressful for the salamander that should be handled very carefully.

Since salamanders almost have a unique abdominal pattern as seen in the pictures below (1,2), NINA would like a system or application that takes an input image of an salamander, check the pattern, determines if the salamander are already in the database.

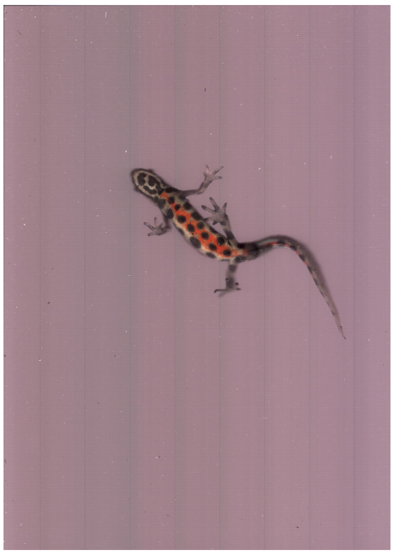


Figure 1: Example of salamander pattern

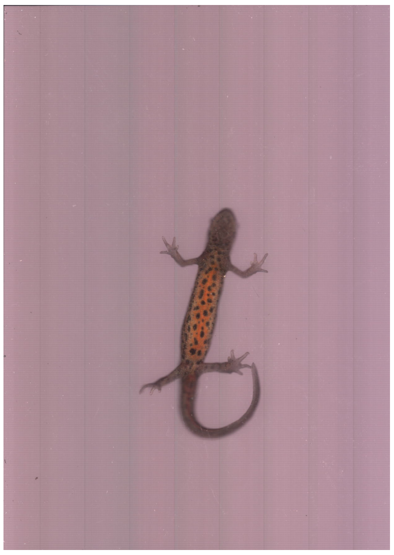


Figure 2: Example of salamander pattern

This system/application will be much quicker to identify and register salamanders to their database. It will also be more gentle and safe for the salamanders and reduce the amount of pit tags needed because they're able to identify the salamander based on pictures only.

1.2 Project goals

Result goals:

- Create a program that can identify salamanders based on pattern recognition and with the highest accuracy that's possible.
- Reduce the amount of pit tags in salamanders

Effect goals:

- Make it easier for NINA to identify salamanders using pictures only
- Faster identification

1.3 Project frames

NINA has not set any limitations on this project. See section 2.2 for our own limitations on the project.

2 Scope

2.1 Problem area

As the salamander is a red listed species, it is crucial to know how it's population is developing. When measures are taken to increase the salamander population, the development of the population gives a good indicator for whether the actions had a positive impact. In order to estimate the population the capture-mark-recapture method is used. In order to calculate the development of the population we look at the ratio between the individuals that have been caught previously and the ones that have not. By using the Lincoln-Peterson method we can then estimate the population.

- N is the total number of salamanders in the population.
- n is the number of salamanders that were marked on the first visit.
- K is the number of salamanders that were captured on the second visit.
- k is the number of recaptured salamanders that were marked.

If we assume that the population is closed, meaning that no individuals die or emigrate between the visits, the ratios of the captured individuals in relation to the total population should be equal to the ratio between the individuals that have a mark on recapture and those that don't.

$$\frac{N}{n} = \frac{K}{k} \tag{1}$$

If we rearrange, we see that we can estimate the total population $\hat{N}$ using the following equation:

$$\hat{N} = \frac{Kn}{k} \tag{2}$$

2.2 Limitations

- Our project does not provide the infrastructure and hardware equipment used for capturing salamanders or taking photographs.
- We are only going to identify the species Northern crested newt, also called the great crested newt (triturus cristatus), and eventually the smooth newt (lissotriton vulgaris).
- We will not take any pictures ourselves and are only going to use the data provided by NINA.

2.3 Task description

NINA already has a solution for photographic identification called AmphIdent [1]. The software works by letting the user mark the boundaries of a salamander and first straightens the image of the salamander before applying pattern recognition. AmphIdent solves the same problem as we intend to do with this project, the goal however is to see wether we can do better than AmphIdent. There are mainly 2 areas where we could be better:

- Higher accuracy
- Reduced amount of manual work in selecting and straightening the the belly pattern.

3 Project organization

3.1 Responsibilities and roles

- Project manager: Jørgen Bakløkken
- Bachelor students: Felix Schoeler, Hugo Nørholm and Jørgen Bakløkken
- Supervisor: Sony George
- Co-supervisor: Marius Pedersen
- NINA representative: Børre Dervo

[1] `https://web.archive.org/web/20190128202817/http://www.amphident.de/en/`

3.2 Routines and group rules

After every meeting there should be clear tasks that everyone has to accomplish by a set due date. In every meeting we look at the task every participant was supposed to complete.

In addition to the minutes of meeting, a weekly status is written where the current tasks in progress that week are recorded.

If a person cannot show up for a meeting, he should try to inform the other meeting participants at least 2 hours in advance. If a person does not show up or two times no actions will be taken. If however meetings are dropped regularly, or there is a obvious lack of work invested by a person, he should receive a warning. If there still is no improvement other penalty measures, such as kicking from the group may be considered.

4 Planning, follow-up and reporting

4.1 Choice of system development model

In our project it is going to be critical to use an agile development model instead of a plan driven one. This is primarily because this project is very flexible, it has one primary goal which is to be able to find a salamander in the database based on an image, because of this we will start by developing code that will work on images with clear chest patterns, and then if we have time work on methods that will allow for a wide array of image quality to be used. We also do not really know how long it will take to implement these methods making it hard to preplan every aspect of the project. In addition the fact that we are a small team of three developers will make easy to implement an agile model. The three agile models we considered for this project were: Extreme programming (XP), Kanban and Scrum

XP might not be the best for our team as it has very strict rules on how you are supposed to work with the project. It demands short cycles with regular meetings with the customer something that might be difficult for us as we do not work in the same area. It also favors highest priority first work order to ensure the best possible product. This is something we will kind of be using as we will start with core functionality and work our way out from there.

Kanban is very simple as all it is is a visualisation of the workflow where you have backlog of tasks on one end of the board and then the tasks make their way through development to the other side of the board where you have the completed tasks. Kanban in addition to giving you a better overview of your tasks is supposed to help minimize how many wip tasks you have at any given time, which in turn makes the project more manageable. Kanban also tells you to measure time spent on each task so that you can use that information to make your development more efficient.

Scrum very much revolves around the delegation of roles, you have the product owner essentially the customer, the scrum master, one for each dev team who is in charge of managing sprints and meeting and dev team members. Scrum works well even with just one dev team but once again our customer is not on-site so a slight downside. Scrum also relies heavily on meetings in the dev team to increase productivity one before each sprint, one short meeting, and a sprint review.

While the productivity increase in Scrum and XP are very enticing the nature of the project as well as a consensus around how we work as a group makes having so many meetings and strict rules about how we work seem a bit redundant. We will derfor use a more kanban style development method where we have a kanban board with all the current tasks. And regular meetings to discuss what functions we should as the project develops.

4.2 Main division of the project

The project will begin with us getting acquainted with the database and the pictures in the database. We will start by comparing the clearer images and when we get that to work we will try to make the program work with more varying image quality.

Describing a timeline for this project is kind of difficult as we are very unsure about how much we will be able to do as this isn't a case of implementing existing technology in a new way we are creating a new algorithm for comparing images of bio metrics. and the entire project rests on this core functionality. We will write a report on the project no matter the outcome, that is why it is important that we document all of our work and write what we tried if it worked and if it didn't and more importantly what we learnt.

4.3 Plan for status meetings and decisions

We have bi weekly meetings planned with our supervisors where we will review our progress and ask for input. We will try to make it so that tasks are completed in cycles around these meetings so that that can be more helpful as we can discuss newly implemented functionality or in some cases tings that do not function.

5 Quality assurance

5.1 Standard tools

- Latex will be used to write the pre-project and project report as well as any other text based documents needed in the project. Due to its configurability, which will enable us to create documents that look just like we want them.
- Visio will be our go to program for creating figures, while Visio might be a tad bit powerful of a tool for what we require in this project we have access to it through NTNU and we have previous experience with the program.
- We will have a bitbucket-repository to store our code in through all of the project as we also have access to this through NTNU and it is something we are very used to from previous school projects. In addition will we be using GitKraken to manage the repository as it makes handling branches and commits more manageable.
- There is also a software that straightens the salamander and highlights the chest patter we will try to use this software at least initially to standardize the images.

5.2 Documentation

5.2.1 Work process

All work should be documented in one way or another, for example meetings are recorded by writing a small report after the meeting and coding will be recorded through the commit history. It is therefore imperative that you commit regularly with good commit messages so that we will be able to to reflect on it later. When it comes to logging time spent on research, we could use a timer to record the time and write our findings, but that seems a bit redundant so we are not going to be recording that in detail, however we are saving all relative material for future use.

5.2.2 Meetings

There will be reports of all meetings which will be included in the final report these will help make everyone remember what was discussed and what was decided. In addition if a member is absent from a meeting the reports are crucial in keeping them up to date.

5.2.3 Code

Well commented code is very important not just for the client, but also to make sure that everyone in the work team understands your intentions and your way of thinking. It is therefore required to describe what each function does even simple ones and more complex code should have smaller comments inside the function to help understand what it is doing if needed. The main comment should be on the lines above where the function starts.

5.2.4 End product

Whatever the end product ends up being we will create manual/wiki so that the user should be able to use the program properly. In addition it is quite likely that the finished product wont be very user friendly making a manual that much more important.

5.2.5 Testing

We will make unit tests for every function where it makes sense to have them. This is important in case there is a need to change the function to make sure that it will retain its use.

5.3 Risk analysis

To analyse risk we imagine a scenario give it a probability on a scale from 1-5, where 1 is nearly impossible and 5 is almost certain. And an impact rating from 1-5, where 1 is negligible and 5 is disastrous. Then we calculate our risk by multiplying. this works well as when the probability and impact of a scenario goes up the risk increases exponentially. The risk then comes to a scale of 1-25, the higher a risk is on this scale the more important it is to have good plan to counter it. In our project there probably won't be many if any very high risks to consider as it does not carry much risk in failure.

- Scenario: One of the team members gets sick, and is unable to attend for a short period.
  Probability: 4 * Impact: 2 = Risk: 8
  Countermeasure: Write reports of meetings and have options for when someone cannot meet in person.

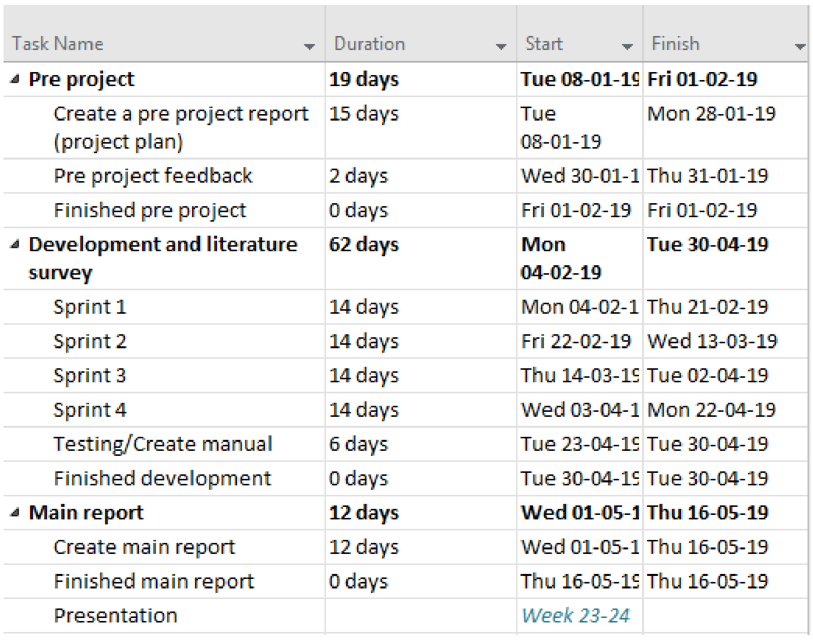
- Scenario: A team member becomes critically injured or sick resulting in him being unable to continue being a part of the project
  Probability: 1 * Impact: 5 = Risk: 5
  There isn't much to be done about this scenario, but to continue the project with the remaining two members

- Scenario: The program ends up only working on really clear or edited images
  Probability: 3 * Impact: 2 = Risk: 6
  Countermeasure: If this is the case then we must give clear instruction of the limitations of the program to allow the customer to use it.

- Scenario: We don't have enough images or enough usable to images to test the program, which leads to difficulty with testing the program.
  Probability: 3 * Impact: 4 = Risk: 12
  Countermeasure: We don't have any way to get more images of Norwegian salamanders until next summer. If this happens to be the case we will just have to make the best of what we have and ssee how it turns out this summer.

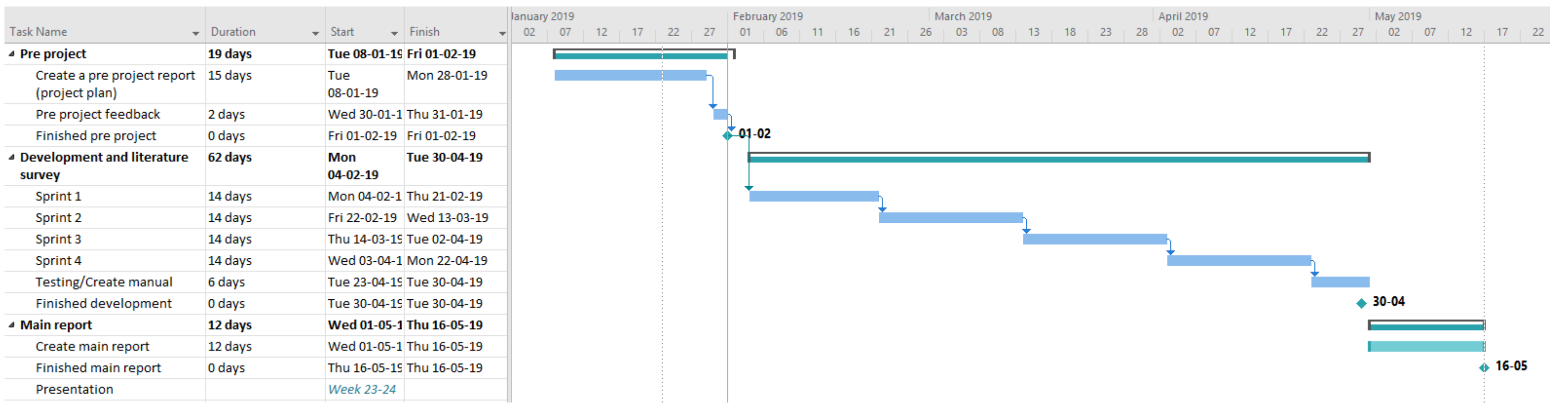
6 Plan for execution

6.1 Gantt schema



Task Name	Duration	Start	Finish
Pre project	**19 days**	**Tue 08-01-19**	**Fri 01-02-19**
Create a pre project report (project plan)	15 days	Tue 08-01-19	Mon 28-01-19
Pre project feedback	2 days	Wed 30-01-1	Thu 31-01-19
Finished pre project	0 days	Fri 01-02-19	Fri 01-02-19
Development and literature survey	**62 days**	**Mon 04-02-19**	**Tue 30-04-19**
Sprint 1	14 days	Mon 04-02-1	Thu 21-02-19
Sprint 2	14 days	Fri 22-02-19	Wed 13-03-19
Sprint 3	14 days	Thu 14-03-19	Tue 02-04-19
Sprint 4	14 days	Wed 03-04-1	Mon 22-04-19
Testing/Create manual	6 days	Tue 23-04-19	Tue 30-04-19
Finished development	0 days	Tue 30-04-19	Tue 30-04-19
Main report	**12 days**	**Wed 01-05-1**	**Thu 16-05-19**
Create main report	12 days	Wed 01-05-1	Thu 16-05-19
Finished main report	0 days	Thu 16-05-19	Thu 16-05-19
Presentation		*Week 23-24*	

Figure 3: Important dates for the gantt schema



Task Name	Duration	Start	Finish
Pre project	**19 days**	**Tue 08-01-19**	**Fri 01-02-19**
Create a pre project report (project plan)	15 days	Tue 08-01-19	Mon 28-01-19
Pre project feedback	2 days	Wed 30-01-1	Thu 31-01-19
Finished pre project	0 days	Fri 01-02-19	Fri 01-02-19
Development and literature survey	**62 days**	**Mon 04-02-19**	**Tue 30-04-19**
Sprint 1	14 days	Mon 04-02-1	Thu 21-02-19
Sprint 2	14 days	Fri 22-02-19	Wed 13-03-19
Sprint 3	14 days	Thu 14-03-19	Tue 02-04-19
Sprint 4	14 days	Wed 03-04-1	Mon 22-04-19
Testing/Create manual	6 days	Tue 23-04-19	Tue 30-04-19
Finished development	0 days	Tue 30-04-19	Tue 30-04-19
Main report	**12 days**	**Wed 01-05-1**	**Thu 16-05-19**
Create main report	12 days	Wed 01-05-1	Thu 16-05-19
Finished main report	0 days	Thu 16-05-19	Thu 16-05-19
Presentation		*Week 23-24*	

Figure 4: Gantt-diagram

C Meeting Logs

You should include in the Appendix a log of your meeting.

C.1 Temporal record of meetings

1st meeting with supervisors

Date: 08.01.2019
Agenda: First meeting with supervisors, startup
Place: NTNU Gjøvik campus
Duration: 1 hour
Meeting content:
Meeting the supervisors, different information about the bachelor thesis

2nd meeting

Date: 18.01.2019
Agenda: Meeting with NINA (Børre)
Place: NTNU Gjøvik campus
Duration: 1 hour
Meeting content:
Meeting with Børre Dervo. Explained about the background for the project, how they catch the salamanders, why do they want this type of software and explained how it should work (ideally). Signed project agreement.

3rd meeting

Date: 21.01.2019
Agenda: Meeting with supervisor (Marius)
Place: NTNU Gjøvik campus
Duration: 1 hour
Meeting content:
Some ideas about what to include in the main report:

- Important to write about the motivation behind this project.
- Include a chapter about previous projects on salamander identification.
- Try the software NINA are using today to straighten and segment the salamander.
- We should focus on the identification part. Let NINA's software do the segmentation?
- 25 hours per week per person. Roughly 500 hours per person.
- Majority decides since we're 3 on the group.

Meetings with NINA : Meeting to get the licence to the software they're using today(not needed if we can use the trial version).
Meeting when we have something that works?.

Tasks:

Felix:

- 3 Project organization
- 2 Scope - Our solution

Jørgen:

- 1 Goals and frames - Information from NINA
- 6 Plan for execution

Hugo:

- 4 Planning, follow-up and execution
- 5 Quality assurance

Should be finished with our tasks within Monday 28.January.
29.January - read through the project plan to check if there's something that needs to be changed.
30.January - send the project plan to Sony and Marius for feedback.

4th meeting with supervisors

Date: 04.02.2019
Agenda: Meeting with supervisor (Sony)
Place: NTNU Gjøvik campus
Duration: 1 hour
Meeting content:
Status so far. Talked about our previous meetings(we will now share our document with Sony and Marius so they can read the meeting notes etc.).
Discussion about how we should proceed.
What should we do in sprint 1:

- continue literature survey
- create a plan for the sprints
- test the software NINA are using today. With different images (some with preprocessing to check the accuracy.)
- Try to find other projects/code online.

Status meeting with group members

Date: 11.02.2019
Agenda: Meeting with group members
Place: Online (Discord)
Duration: 1 hour
Meeting content:

- Images have to be straightened before pattern recognition.
- It's adventagous to covert the images to binary.

- AmpIdent does:
  1. Convert image to a small standard size.
  2. Convert images to binary.
- Tasks

5th meeting with supervisors

Date: 18.02.2019
Agenda: Meeting with supervisor (Sony)
Place: NTNU Gjøvik campus
Duration: 1 hour
Meeting content:
Status so far:

- Existing paper and software review. Hugo reviewed 4 different softwares. Commonalities include that straightening, cropping, rotation and tresholding are applied as preprocessing steps. The difference is in the matching algorithm. Some use feature based methods while others use pixel based comparison methods.
- Felix converted Access database to proper folder structure with unique id for every salamander.
- Jørgen found tresholding methods that we can use.
- Hugo also worked on cropping and rotating of an image based on the four feet as reference points.

Next Sprint

- Felix - straightening salamander image using belly line. Marius suggested to use something similar to control point registration (MatLAB function CPSelect.)
- Jørgen - Figuring out feet reference points after straightening for cropping and rotation.
- Hugo - Find applicable matching algorithm
- Future work chapter should be added where we outline what other steps can be done after the bachelor thesis: |. Measuring length of salamander. 2. Determining the salamanders species.

Status meeting with group members

Date: 25.02.2019
Agenda: Meeting with group members
Place: Online (Discord)
Meeting content:

Status meeting with group members

Duration: 1 hour
Date: 14.03.2019
Agenda: Meeting with group members(Jørgen and Hugo)
Place: NTNU
Duration: 1 hour

Meeting content: Discussion about our code and how we should proceed.

Status meeting with supervisors

Date: 18.03.2019
Agenda: Meeting with group members and supervisors
Place: NTNU
Duration: 1 hour
Meeting content: Status on feature extraction and the straightening.
Discussion about the final report (finished table of contents mid april?), a lot of theory in this project so we should start working on the final report in april.

Things to do/keep working on:

- Continue working on the straightening
- Try DSIFT on the original pictures, thresholded pictures and pictures with different lighting.
- Find a smart solution to store the features in some sort of a database.

Status meeting with supervisors

Date: 25.03.2019
Agenda: Meeting with group members and supervisors
Place: NTNU
Duration: 1 hour
Meeting content: Individual update from group members. Discussion about table of contents.

Status meeting with supervisors

Date: 03.04.2019
Agenda: Meeting with group members and supervisors
Place: NTNU
Duration: 1 hour
Meeting content: Update from group members. Discussion about the report (mainly table of contents.)

Status meeting with supervisors

Date: 12.04.2019
Agenda: Meeting with group members and supervisors
Place: NTNU
Duration: 1 hour
Meeting content: Discussion about the report.
Things we could consider to include:

- Show how much faster/better our solution is compared to other solutions
- Introduction - floatchart about what we're going through in the thesis (this part-part is discussed in this and this chapter)
- Include a subsection about other literature (one chapter or split it into other subsec-

tions) - explain why we did this. Intended audience?

Status meeting with supervisors

Date: 23.04.2019
Agenda: Meeting with group members and supervisors
Place: NTNU
Duration: 1 hour
Meeting content: Meeting notes: Discussion about the next 4 weeks.
Include what we want to do in the start of the report (project goals).
More testing - false matching rate etc.
In discussion - image quality etc. (sharpness and contrast).
Include image requirements in report.

Status meeting with group members

Date: 23.04.2019
Agenda: Meeting with group members
Place: NTNU
Duration: 6 hours
Meeting content: Coding.

Status meeting with group members

Date: 26.04.2019
Agenda: Meeting with group memberss
Place: Discord
Duration: 1 hour
Meeting content:

Status meeting with group members

Date: 29.04.2019
Agenda: Meeting with group members
Place: Discord
Duration: 1 hour
Meeting content:

Status meeting with supervisors

Date: 07.05.2019
Agenda: Meeting with supervisors
Place: Gjøvik
Duration: 1 hour
Meeting content: Discussion about what to include in the report.
Feedback from supervisors.

Status meeting with group members

Date: 09.05.2019
Agenda: Meeting with group members
Place: Discord

Duration: 1 hour
Meeting content: Update from Felix about the belly identification.
Discussion about the report.

Status meeting with group members

Date: 11.05.2019
Agenda: Meeting with group members
Place: Discord
Duration: 1 hour
Meeting content: Report discussion.

Status meeting with supervisors

Date: 13.05.2019
Agenda: Meeting with supervisors
Place: Gjøvik
Duration: 1 hour
Meeting content: Feedback on the current report from supervisors.