GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Bachelor's Thesis

Cable Detection in Automated Dissassembly Environment using Deep Learning

Kabelerkennung in Automatisierter Demontageumgebung mittels Deep Learning

prepared by

Michael Olliges

from Vinnen

at the Third Institute of Physics

Thesis period:	2nd August 2019 until 25th October 2019
First referee:	Dr. Minija Tamošiūnaitė
Second referee:	Prof. Dr. Florentin Wörgötter

Abstract

Cables are essential components of any device in the electronic waste sector. For several years disassembly processes in this sector got automated step by step. However, most of these automation's were device-specific and not usable on a wide range[1]. An advantage came about as neural networks improved in image recognition. The recent improvements marked the starting point for many research groups to focus on disassembly lines that can handle multiple devices.
This bachelor thesis aims to address a fundamental problem in such disassembly processes: **Cable-Detection**.
Consider the task of disassembling a DVD-Player which still includes usable components. Cables connect most of these components for information and energy exchange. Therefore one main goal before removing individual components is to cut the cables between the components to ensure their safe removal. Inevitably this task needs a precise knowledge about the position of every single cable.
For a disassembly robot, a strategy that is worth pursuing is to locate the cables processing an RGB image. An RGB image is a lightweight solution with high standards regarding the improvement over the last year.
This Bachelor thesis presents a model for cable detection. A state-of-the-art instance segmentation model Mask R-CNN [2], published in 2018, is trained.
The presented work is public on GitHub[3] and the used Dataset on kaggle[4], to make this global concerning topic available for the public.

Keywords: Deep Convolutional Neuronal Networks, Mask R-CNN, Cable-Detection, Disassembly, E-Waste

[1] https://www.apple.com/environment/pdf/Apple_Environmental_Responsibility_Report_2018.pdf
[2] https://github.com/matterport/Mask_RCNN
[3] https://github.com/DerOzean/Cable-Detection-in-Automated-Dissassembly-Environment-using-Deep-Learn
[4] https://www.kaggle.com/zeuscasio/cable-dataset-annotation

Contents

Nomenclature

Abbreviations

Abbreviation	Meaning
2D	Two-Dimensional
FCN	Fully Convolutional Network
FPN	Featured Pyramid Network
R-	Region-
CNN	Convolutional Neural Network
IoU	Intersection over Union
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
COCO	Common Objects in COntext
LR	Learning Rate
ResNet	Residual neural Network
SGD	Stochastic Gradient Descent
Fig	Figure
Sec	Section
AI	Artificial Intelligence
NASA	National Aeronautics and Space Administration
Event	is used synonyms for Ground Truth

Declaration of Ownership

I, Michael Olliges, declare that this project titled, 'Cable Detection in Automated Disassembly Environment using Deep Learning' and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a bachelor degree at the Georg-August-University of Göttingen.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

1. Introduction

In a report concerning e-waste released by the United Nations University[1], global electronic waste has reached a record high of 41.8 million tons in 2014 and has since grown according to estimates[2](See Fig: 1.1). Most of the waste ($\sim$ 60%) are small and large types of equipment for homes like vacuum-cleaners, TV's, washing machines and video cameras. Therefore the main fraction of e-waste starts its life in ordinary homes due to a disposable mentality in developed countries.

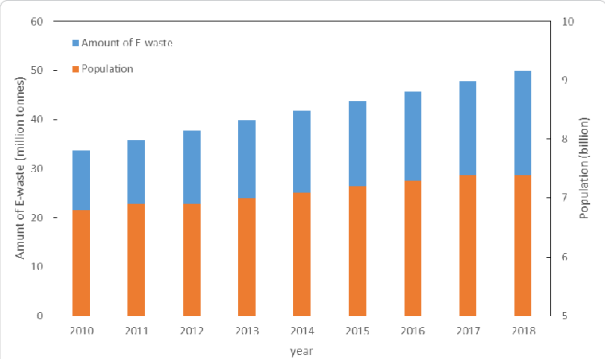

Figure 1.1.:
Global e-waste generation from 2010 to 2018.[3]

While e-waste is a source of rare metals like iron, gold, aluminum, and copper which make up more than 60% of the total weight of e-waste, only $\sim \frac{1}{5}$ of the total e-waste gets recycled probably. Correspondingly shipping e-wast to emerging countries for disassembling or burning it due to wrong handling makes up a total of $\sim$ 80%. This method releases toxic smoke to the air and heavy metals to the ground, which can impact nature for decades[4]. Accordingly, to save the environment for the coming generations, it is crucial to think about new concepts which can help to recycle e-waste.
Thanks to the considerable portion of rare metals, disassembly of e-waste is already a multi-million dollar business that can become even more lucrative in the future. The most astonishing benefit of efficient recycling is that it can be even cheaper than conventional methods for producing most metals[5]. Therefore all there is left to do is to find an efficient method of disassembling old devices and extracting the valuable metals.
A promising approach are deep neural networks which benefit from the growing usability of object detection and semantic segmentation over the last years. These advantages can help to detect parts of devices that are still usable and separates the remaining parts specific to their ingredients.
The new advances in this area which give the opportunity of fast and proper detection

[1] https://unu.edu/news/news/ewaste-2014-unu-report.html
[2] https://www.sciencedirect.com/science/article/pii/S0921344917300290
[3] https://www.researchgate.net/publication/289976734_Literature_Review_of_the_Hydrometallurgical_Recycling_of_Printed_Circuit_Boards_PCBs
[4] https://envibrary.com/electronic-pollution/
[5] http://www.diva-portal.org/smash/get/diva2:647881/FULLTEXT01.pdf

were made possible by the improvement of baseline systems such as Fast[1]/Faster[2] R-CNN and Featured Pyramid Network [3]. Over the last decade, these conceptually intuitive methods became flexible and robust over time, and also made fast training for object segmentation possible.
The ambition for this thesis is the detection of cables in old electronic devices to make a step to automated disassembly of e-waste. For accomplishing this goal, the used framework[4] is a general framework for object instance segmentation based on Faster R-CNN and FPN.
Mask R-CNN trains a convolution network for semantic image segmentation. Thereby the displayed information (pixel) is used to conclude the events on the images. The model then generates a collection of local segmentation masks describing each of the region recognized in the image.
The used Framework is open source and well documented on Git Hub[6]. Accordingly Mask R-CNN is a state of the art network and improves continuously due to the availability on the Internet.

2. Related Work

Automated disassembly of e-waste gained recognition in society over the last decades[1]. This focus on recycling nowadays is a consequence of the many expensive raw materials in electronic devices and is a consequence of the rising prices of rare metals and the increasing public spotlight on environment damaging practices in obtaining these materials[2].
Researchers also became interested in e-waste because it has advantages compared to regular waste. The extensive usage of screws which comply with norms and repeating structures made these topics tangible. Even if automation of a few processes were successful in the past [5] some hurdles prevent a breakthrough so far. For example, the wide variety of components and the freedom of design is a big problem for disassembly.
One of the earliest attempts at automated disassembly of electronic waste goes back to 1998. In paper [6] a scheme is suggested on how to handle disassembly of e-waste and what has to be done to make a semi-automated disassembly work. However, there are no clear solutions to the problem at hand, and the author mentions that successful solutions only work with select devices. Therefore, the author suggested setting up collection points for each type of device. This suggestion opposes the premise of this thesis. The overall goal is to disassemble every kind of device with the same robot. The problems described in the paper have not changed a lot over time, and a satisfactory solution is still not found.

[6] https://github.com/matterport/Mask_RCNN
[1] https://www.forbes.com/sites/vianneyvaute/2018/10/29/recycling-is-not-the-answer-to-the-e-waste-c
#2c848fe97381
[2] https://www.itu.int/en/ITU-D/Climate-Change/Documents/GEM%202017/Global-E-waste%
20Monitor%202017%20.pdf

One of the earliest attempts to detect wires started in 2001. NASA research suggested

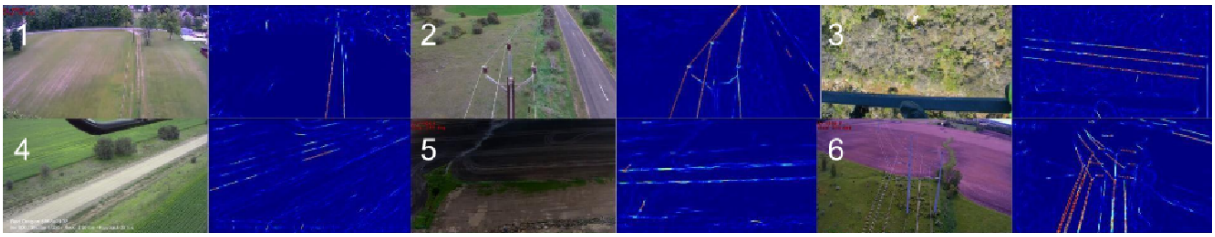

Figure 2.1.: Heatmap results of [7]. The more the color turns red, the more secure the network is for a wire[3].

extracting an edge map using the Steger algorithm. In paper [8], NASA researched a way to detect cables for low altitude helicopter flight. The risk of electric cables during a flight is not to be underestimated and therefore lead to this research. While the human eye is not sensitive to black wires on a textured background, a computer model can easily detect lines.

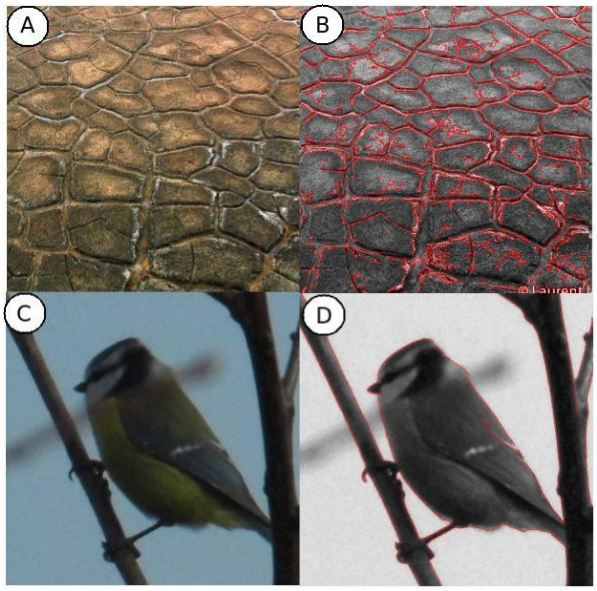

Figure 2.2.: **A + B:** Edge detection applied to a ice-wedges image[4]. **C + D:** Edge detection applied to a bluetits image.

On the other hand, the problem with computer vision is that it cannot distinguish between two events without a complex strategy. This strategy is nowadays build up with a neural network and not hard-coded. The article explains the basics of cable detection and what is reachable in short terms.
A similar idea was persued in 2017 by the Carnegie Mellon University in Pittsburg. In paper [7] a strategy is presented that allows automated detection of power lines using

[3] https://www.semanticscholar.org/paper/Wire-detection-using-synthetic-data-and-dilated-for-Madaan-I
4041cd4993409d851b0c3db4bc799324efbe62e6
[4] http://www.alaskaphotoworld.com/alaska365/2012/10/05/ice-wedge-polygons/

2D-images/videos. It is supposed to generate synthetic data in order to train a deep neural network. This network is used to generate semantic segmentation masks around the events just like planned for this thesis. The proposed network is specified on an image resolution of $460x640$ pixels to make a real-time detection possible. In image 2.1 some results of the paper are represented. The main problem is that straight lines can be confused with wires. The same problem occurs here were some roads (See Fig: 2.1 4) are labeled as wires. Never the less the results are reliable, and misinterpretations occur rarely. The used framework Mask R-CNN is an extensive groundwork, for instance, segmentation masks and therefore used in many areas. In paper [9] the framework is used to detect ice wedges and categorize them into subgroups (See Fig: 2.3). Moreover, a modification of the same framework detecting birds and categorize them into species, in paper [10]. Both of these papers present the most common use of Mask R-CNN in which the goal is the detection of circles like polygons with a huge inner area. Figure 2.3 $Left$ shows this specific usage of Mask RCNN. The figure shows a detection in which nearly the whole image contains ground truths of circle-like ice wedges. The interesting point is that edge detecting would give a similar result in both cases. For verifying how Mask R-CNN works, an edge-detection[11] was performed on a bluetit and an ice-wedges image (See Fig: 2.2 A and C). In the resulting images (See Fig: 2.2 B and D) can be seen that edge detection already generates very similar results compared to the results presented in the paper (See Fig: 2.3). Therefore Mask RCNN is in these cases mostly used to find closed circles and interpret the color spectrum inside these circles.

An additional problem with many previous works is the usage of Mask RCNN on images where the events are in the center and cover most of the area. These are significant differences compared to a cable which can lay anywhere within the area and may cover less than $\sim$ 5% of the total area. Therefore, this thesis reviews how well Mask R-CNN performs on structures without any significant texture within there area and structures which cover only a fraction of the total area.

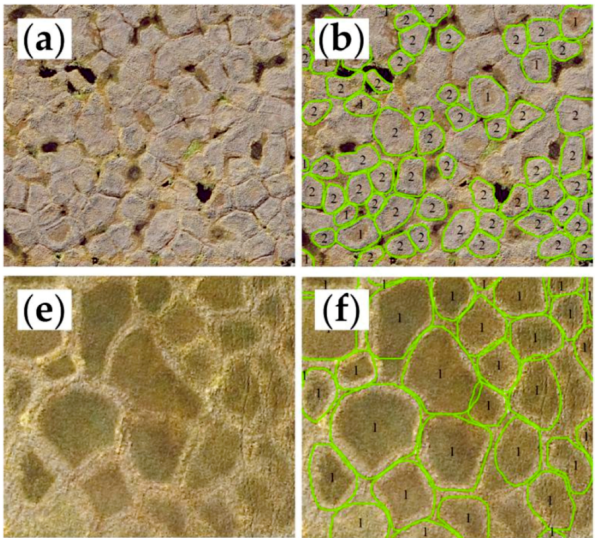

Figure 2.3.: Characterization of Arctic Ice-Wedge Polygons via Mask R-CNN[5]

[5] https://www.mdpi.com/2072-4292/10/9/1487

3. Background/Theory

3.1. Machine Learning

3.1.1. History

Machine learning is a preamble for the artificial generation of knowledge from experience. The idea is that there are generic algorithms that can say something about the data without writing a specific code. Instead of writing code, data is faded to the algorithm so the machine can build up its strategy. Therefore Machine learning enables not just to learn examples but to recognize patterns in the training data. So the final program can evaluate new data due to experience with similar structures[1].
The rise of neural networks began in the early 1950*s* when the question "Can machines think like humans?" was formulated. Due to the fast progress in storage and computing power in this decade, Marvin Minsky and Dean Edmonds tried to answers the question and built the first artificial neural network. They tried to simulate an organic brain with stochastic approaches lend from mathematical-concepts[12].
After the first tests in the 1950*s*, the expectations were high. Therefore many became disillusioned when breakthroughs did not happen in the first years. This disillusion led to a decrease in funding during the 1960*s* and 1970*s* which slowed the progress until the late 1980*s*.[2]
In the late 1980*s* people renew trust in neural networks which now rely on a new model called backpropagation, which is still used today in many algorithms. Thanks to this and new funding, neural networks could produce in a short period results no one dreamed about earlier. For example, the neural network "Deep Blue" beat world chess champion Garry Kasparov in 1996 [3].
Nowadays, neural networks still catch the interest of the public and discover new areas for its usability. In 2016 "Alpha Go" a neural network based on a Monte Carlo tree search algorithm to find moves was able to beat the number one player Ke Jie in the game Go[4].
Even today neural networks are a powerful tool in many branches of science, like medicine, biology, physics, and computer science.
Most of these branches use deep neural networks for image recognition and try to improve usability in their fields. Thanks to this wide variety of tasks and branches, image recognition became one of the most successful branches in soft computing[13] over the last year.

[1] https://skymind.ai/wiki/neural-network
[2] b.org/deeplearning/2017/02/23/deep_learning_101_part1.html
[3] https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/
[4] https://medium.com/point-nine-news/what-does-alphago-vs-8dadec65aaf

3.1.2. Neural Networks

Neural networks are versatile for classification tasks. One framework can accomplish multiple tasks without changing the underlying code. Finding the right framework for one's task is challenging. There are two different kinds of learning approaches:

1. **Supervised Learning**, generates an output to every given input which is overseen by a teacher.

2. **Unsupervised Learning**, generates an output based on the experience of the network with similar inputs.

Because the expected result is known, a supervised network like Mask RCNN is a promising approach.
The General Concept of a neural network is to process a given input through multiple layers. For classification of images, every pixel gets represented by a Value in an array. This Array is the zeroth layer of the network (See Fig: 3.1). Afterward, the pixels proceed to the next layer and get assigned to a neuron based on their value and position. In these neurons, a function calculates the new value based on the input and addresses the input to the next layer. This procedure is going on for every layer until the last one, which assigns the input to a possible outcome (cable, no cable)[5].

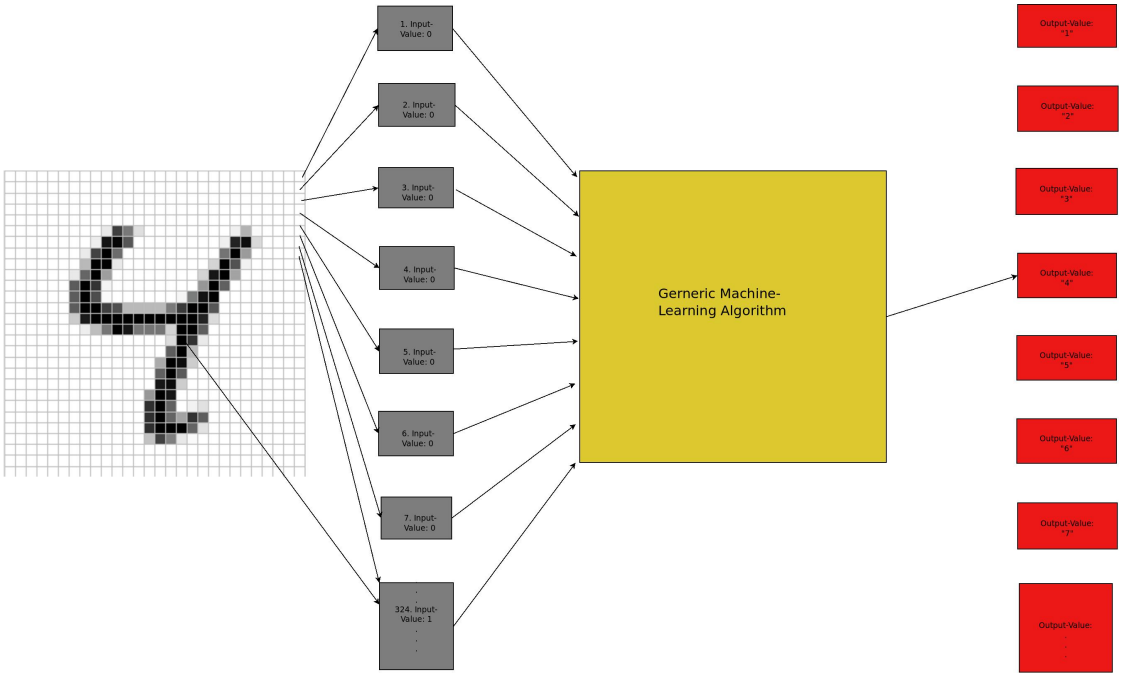

Figure 3.1.: Schematic sketch of subgroups in artificial intelligence.

[5]https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

3.1.3. Deep Neural Network

Deep learning is one of the most significant subgroups of Machine learning (See Fig: 3.2) due to a wide variety of application areas and fast training. Compared to shallow networks which have 1 or 2 layers a deep neural network has a depth of typically 50 to 150 layers.

The advantage of building a deep neural network is the higher complexity. This complexity allows the networks to recognize manifold structures.
Biological neurons inspired these basic structures, and just like them, not every layer communicates with every other. The first layer is always giving its calculated value to the second layer, and so on. The primary reason why deep neural networks are used so often nowadays is the better performance on many problems, primarily image-recognition.
However, this is not the only reason for its success. Deep neural networks make problem-solving easier. Previous techniques only transformed the input in one or two representative spaces. These transformations were mostly high dimensional non-linear projections or decision trees. The significant disadvantage was that humans had to transform the input data manually into a more amenable form before passing them to the network.
Even if the network could train the layers independently, it was always a big deal finding a functional representation for the input. Deep Learning, on the other hand, completely automates this step. Deep neural networks are trained end-to-end in one run and used in the same way.

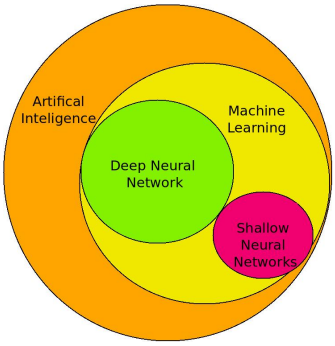

Figure 3.2.: Schematic sketch of the hierarchy between AI, Machine Learning, Deep Learning, and Shallow Learning

3.1.4. Convolutional Neural Networks

A Convolutional Neural Network is a deep neural network architecture specially designed for processing images. CNN's are used in object detection mainly because they can process arrays instead of vectors. In most other neural networks, the input gets represented by number sequences. These sequences are unique for every input data. Therefore a number combination in the beginning has another interpretation than the same number sequence at the end.
On the other hand, CNN's represent pixel in the right order, and repeating combinations can be detected not regarding the position of the event. CNN's are also able to process high dimension inputs. Higher dimensions of the array represent most of the time color channels. Accordingly, most CNN's are processing a 6-Dimensional array (height x width

[6] https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/

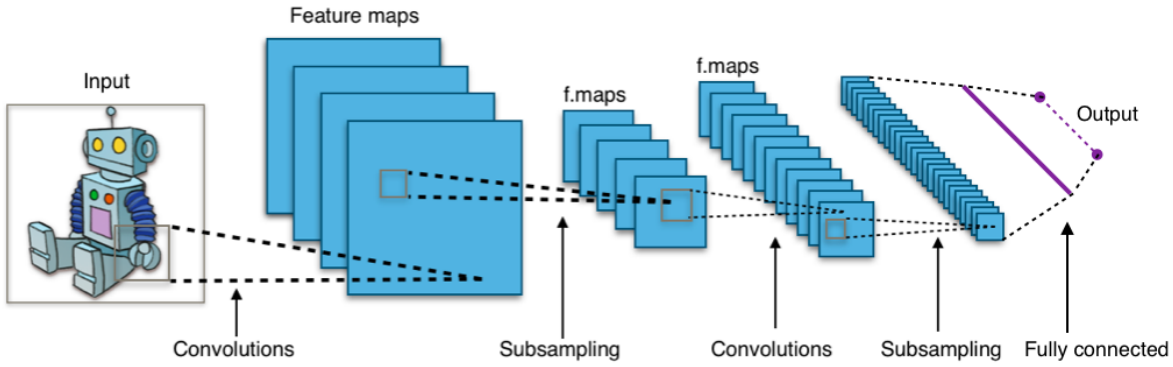

Figure 3.3.: Schematic sketch of processing an image in CNN.[6]

x depth x red x blue x green).

Processing images in CNN works as follows. First, the input image has to proceed through a convolution layer. This layer is usually a $2x2$ or $3x3$ pixel-grid, which calculates a results matrix. For the calculation of the new matrix, the filter moves over each row of the image and uses weights to calculate a compressed version of the original input array. Compression happens when a $2x2$ array is converted to a $1x1$ array.
This compression happens $16 - 32$ times to the original input with different filters. The results are $16 - 32$ result matrices (feature maps) which are all processed to the next step (See Fig.: 3.3). The next step usually is again a convolutional layer with the same properties as the first.
After the first convolutional layers, the results get processed by a pooling layer. The pooling layer again uses a pixel-grid. The pixel, in contrast to the previous layer, do not combine into one due to the weights. Instead it takes the highest value and passes it to the next layer.
In most common CNN's the whole process discussed so far is repeating once more to filter weak signals and compressing the input into handleable array sizes.
After pooling the input is processed by one or multiple fully connected layers. These layers resemble a neural network in which inputs and outputs are in connection with every neuron. For processing, the input gets flattened. That means that all arrays so far have to be restricted to a number sequence. This flattening destroys every position information of the pixel, but position independent object information stays intact.

3.2. Training

Training is an essential part of building a network. During training, the network gets specified to all characteristics of the objects which it later has to detect.
Training usually involves **Backpropagation** nowadays. Backpropagation is a generalization of the $Least - Mean - Squares - Algorithms$ and therefore, it is only practical for supervised learning.
The Basic Algorithm works as follows[14]:

1. A specified input is propagated forward through the network.
2. The network compares the generated output with the correct output. From this comparison, the networks calculate the loss as a measurement of the error
3. In the last step, the error backpropagates through the network layers. During this, the neurons get checked due to the impact on the error. Afterward, the neurons get changed to minimize the loss. The LR restricts the potential impact of the changes during one iteration.

The number of **Steps** is synonymous with the number of total repetitions during one epoch. At the end of an epoch, the network repeats method 3.2 with the collected validation data to check the network on new data.
This procedure garnets an improvement the next time the same input propagates through the network. Due to only slight changes to match the output the network is not fixed to one input and can still detect similar images it has never seen before.
A critical aspect of improving the network during training is the **Learning Rate**.

1. If the LR is too high, the networks can find in short time minimums (See figure: 3.4 c)). The disadvantage is that it will not get to the bottom of the found minimum because the jumps between steps are too big. Additionally, the network will jump between near minimums, which will decrease the quality of the network in later epochs.
2. If the LR is too small, the network will get to the bottom of a minimum over time (See figure: 3.4 a)). The disadvantage is that these minimums are just local ones. The network will slowly propagate in these minimums and cannot improve in later epochs.
3. Decreasing the LR over time can be an advantage for training. The network can make big jumps and find global minimums in the beginning. In later epochs, the network gets to the bottom of minimums with a smaller LR. The only disadvantage can occur when the network focuses a local minimum after the first epochs without taking others into account anymore.
4. Cyclical-LR [15] means to repeat decreasing LR multiple times during one training. Benefits are the fast progress if the network is on a plateau, the capability to change aimed minimum in later training and to find the bottom of a minimum due to the sometimes small LR (See figure: 3.5).

The last parameter discussed in this thesis that has an impact on the training is the used **optimizer** . An optimizer helps to find minimums (or maximums) of an objective function $E(x)$ (Error function). The optimizer must determine an execution plan for a query. This plan includes decisions about the access order for arrays referenced in the query, the join operators, and the accessor methods used for each array. Therefore the right choice

[7] `https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-de`

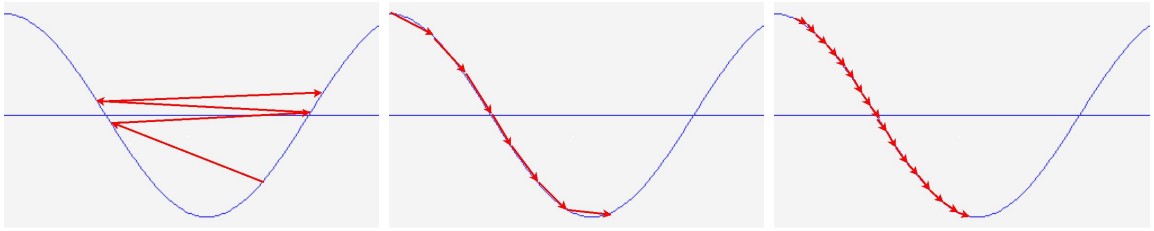

Figure 3.4.: Comparison of constant LR's. a) High LR causes drastic updates that can lead to decreasing quality. b) Good LR nearly misses the minimum c) Low LR leads to many updates until it reaches the minimum

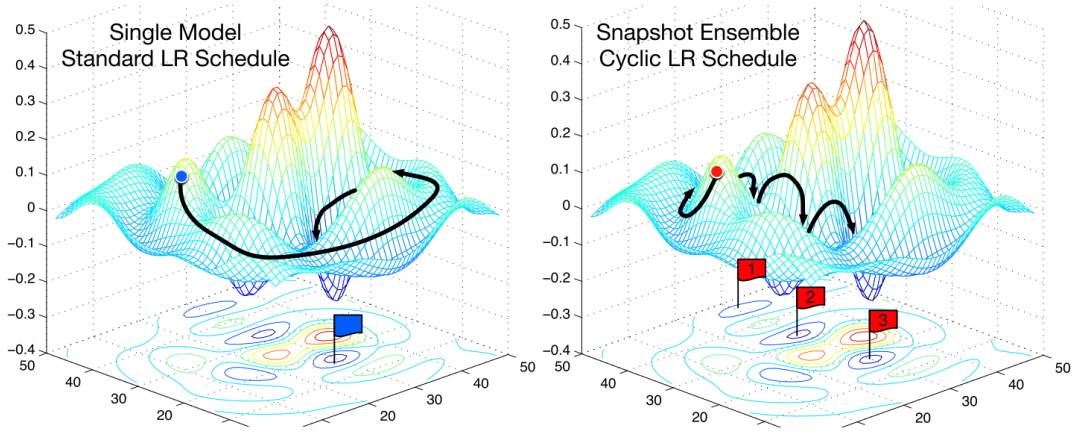

Figure 3.5.: Comparison of different LR approaches. a) Constant LR converges to a minimum until the end of training. b) Cycling LR approaches local minimum and leaves them again[7].

of the optimizer can minimize the loss faster and generate better results.
The paper of Mask R-CNN suggest the SGD-optimizer which includes support for momentum, LR decay, and nesterov. In the deep learning community many suggest that the $ADAM$-optimizer generates better solutions in many cases (Discussed in: [16]). Therefore both optimizers are tested on the problem to check which one is the best for this task.
The basic concept of both optimizers is a gradient descent algorithm. Gradient descent is an algorithm to find the minimum of a multidimensional function.
First, the algorithm takes a small coefficient as a starting point like:

$$\text{coefficient} = 0.0 \tag{3.1}$$

This coefficient gets plugged into the function $E(x)$, which calculates the cost value. In other words, the loss is determent, which represents the quality of the network.
From this cost, the derivative is formed to get the slope of the function ($delta = derivative(cost)$) at a specific point. With the slope, the network knows in which direction the function

decreases/increases, and it can update the coefficient.

$$\text{coefficient} = \text{coefficient} - (\text{LR} \cdot \text{delta}) \tag{3.2}$$

This whole strategy seems straight forward as long the error function is known. Nev-

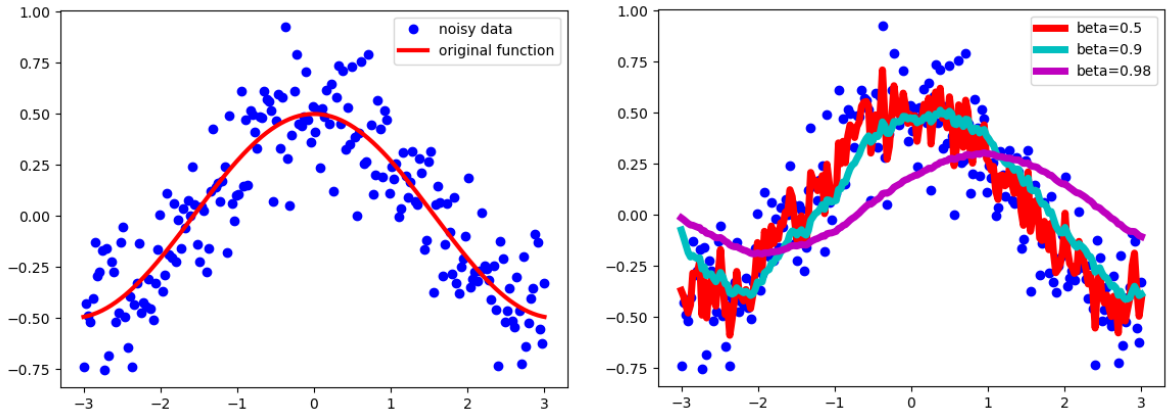

Figure 3.6.: **Left:** Cosine overlaid with noise to generate an example function.
Right: Different values of β compared to how good they fit the cosine[8].

ertheless, during typical training, the error function is not known; otherwise, it would be no problem to calculate the exact minimum in no time (See Fig: 3.6 left). To handle this uncertainty, exponentially weighted average deals with sequences of numbers (S) and form an approximated function (V) [16]:

$$V_t = \beta \cdot V_{t-1} + (1 - \beta) \cdot S_t \tag{3.3}$$

In this equation, β is a parameter chosen between 0 and 1. Figure 3.6 B shows the impact of *beta* on the fit. If β is big, the algorithm creates a smooth function which is not fitting the truth perfectly. If β is small, the algorithm creates a non-smooth function which represents the shown data tidily. Both extremes can be critical. While a non-smooth function can lead on small scales to a wrong slope, a not fitting function leads to a minimum displaced from the true one.
Gradient descent alone can be slow on huge datasets due to its prediction on any instance/image.
In 1951, Herbert Robbins and Sutton Monro invented the basics of **SGD**-optimizer[9], to avoid time-consuming training. The benefit of SGD is that it updates the coefficient after every image has passed through the network. Therefore the benefit of SGD is that it updates less frequently and takes a combination of inputs into account. Therefore It is important to randomize the order of the input data every epoch, to give each data point the same impact on the result.

[8] `https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d`
[9] `https://www.jstor.org/stable/2236626?seq=1#metadata_info_tab_contents`

Adam (Adaptive Moment Estimation) is an adaptive learning rate optimization algorithm, designed specifically for training deep neural networks. It is working at the same base-system as the SGD, but it is additionally computing adaptive learning rates for each parameter. To do so, it has to store the last gradients v_t. This strategy is similar to momentum and keeps an exponentially decaying average of past gradients m_t [17]:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{3.4}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{3.5}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance). The parameters β_1 and β_2 are constant values chosen between 0 and 1.

3.2.1. Loss Function

The loss function is a beneficial tool to predict the quality of a network. After every step, the network gets checked on how well it performed on the training data (See Sec: 3.2). These checks include the processing of one image through the network and afterward the comparison of the output (y') with the ground truth (y).
An intuitive way of getting a loss-value ($L(y, y')$) would be:

$$L(y, y') = \frac{1}{m \cdot n} \sum_{i=1}^{n} \sum_{j=1}^{m} (y_{ij} - y'_{ij})^2 \tag{3.6}$$

In the above equation $m = height$ and $n = width$ of the input image. Where the output value of one pixel (y'_{ij}) (on position $x = i$ and $y = j$) is subtracted from the actual value (y_{ij}) and squared afterward (because negative values are as bad as positive ones). This loss-value gets calculated for every pixel. Then the sum over all pixel is determined and divided by the total number to get the mean.
Mask R-CNN generates the loss slightly different due to the additional aspect of masks. In the paper of Mask R-CNN it is explained that the loss is a multi-task-loss, which is the result of the following equation:

$$L = L_{cls} + L_{box} + L_{mask} \tag{3.7}$$

Here L_{cls} describes the classification loss and L_{box} the bounding-box loss, which are defined similar to equation 3.6 in the Faster R-CNN paper.
L_{mask}, on the other hand, is defined in the Mask R-CNN paper as the average binary cross-entropy loss:

$$L_{mask} = -\frac{1}{m^2} \sum_{1}^{i} \sum_{j}^{m} [y_{ij} \cdot log(y'^{k}_{ij}) + (1 - y_{ij}) \cdot log(1 - y'^{k}_{ij})] \tag{3.8}$$

It is only including k^{th}-mask if the region is associated with the ground truth class k.
The same procedure repeats every epoch for the validation data. Also, the validation-

loss-function is a multi-task-loss:

$$L_{Val} = L_{cls->Val} + L_{box->Val} + L_{mask->Val} \tag{3.9}$$

In this equation, the abbreviations and associated functions are the same as for the training data.

3.2.2. Overfitting

Networks that adapted a dataset too strictly may suffer from overfitting. Overfitting occurs when datasets are too small so the network can save nearly every detail about the data represented. Overfitted networks can detect precisely the objects from the training data but cannot detect similar objects.
Identify overfitting while training can sometimes be difficult without checking the network on new data.

One reliable method is to compare the validation-loss (Equation: 3.9) with the training-loss (Equation: 3.6). If the training-loss decreases continuous, and the validation-loss starts to rise, it is an indicator of overfitting. It is because the network starts to specify on a few images that are not representative for the general input. While this specification starts, the networks get worse on validation data because this data is not known by the network.
There are many strategies to prevent overfitting:

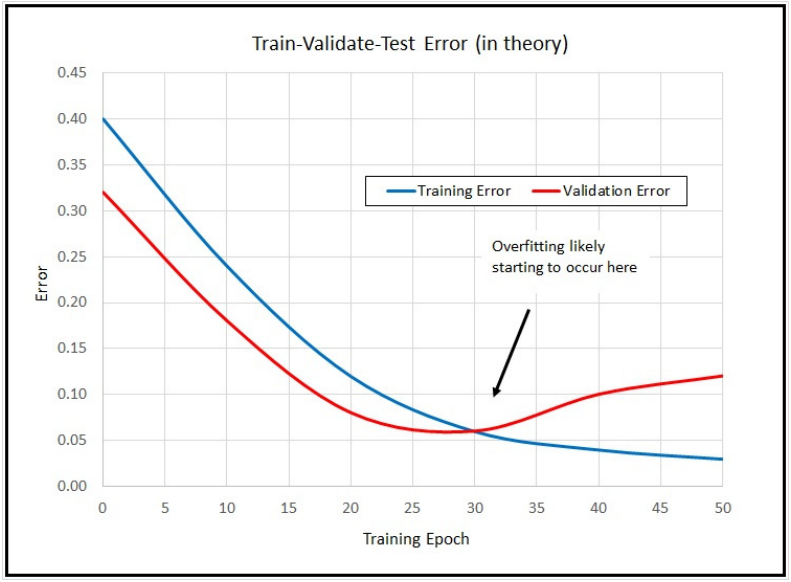

Figure 3.7.: Schematic sketch of a loss function representing an overfitted network. [10]

1. **Collecting more data** is always an excellent strategy to improve a network. It always gives new information to the network which makes it less likely to overfit. Even if it is the most promising strategy, it is sometimes time-consuming and challenging to find more data.

2. **Augmentation** is a strategy where the existing data is modified to generate slightly changed data. Because the network takes position, color, and sharpness into account, a slightly changed image gives new information to a network. These changes

[10] https://jamesmccaffrey.wordpress.com/2018/06/28/neural-network-train-validate-test/#jp-carousel-8772

can include left-right flips, up-down flips, rotations, translations, zoom in and out and changes in colors, contrast, and brightness. After augmentation, nearly every changed image is new to the network, and therefore overfitting is getting less likely. Nevertheless, augmentation is not without danger. A new image generated via augmentation has to represent real examples that could occur in the later usage of the network. For example, a network trained on detecting walking humans would not benefit from up-down flips.

3. Training a **smaller network** is a very intuitive solution. If the network has not enough capacity to save all information about an object it is less likely to overfit. The most significant disadvantage is that a smaller network is also not able to specify that much due to its size. Therefore a small network will not generate as good results as a big one if the object structure is complex.

3.3. ResNet

By designing increasingly deeper networks, it became essential to deal with increasing complexity and expressiveness of networks by adding layers. It was also essential to make deep neuronal networks strictly more expressive rather than just more complex.
To face this hurdles ResNet was invented in 2015 by Microsoft Research Asia [18]. The architecture soon obtained very successful results in the ImageNet[11]- and MS-COCO[12]-competition and is still today one of the most used backbones in Deep Learning.

The idea of ResNet is inspired by pyramidal cells in the cerebral cortex of biological systems. ResNet is propagating some inputs x without passing them through the neurons, just like structures in the cerebral cortex.
In the schematic sketch 3.8, is an input x demonstrated, which is skipping two layers. During the layer skip, neurons do not get completely ignored. As long as the neurons output > 0, the calculated output gets added to the earlier input.
A primary benefit of this strategy is to avoid the **vanishing gradient problem**.
In earlier networks, the vanishing gradient problem was that some neurons could become vanishingly small. These neurons would, in the best case, include an extra layer to pass without significantly changing the output. In the worst case, it could prevent the network from further training due to the multiplication with zero.
The implementation of Mask R-CNN suggests two different backbones (*ResNet*50 and

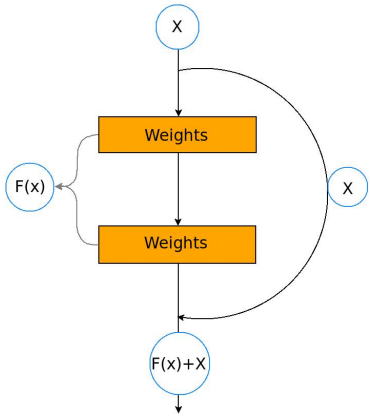

Figure 3.8.: Schematic sketch of ResNet skipping two neurons to improve the training.

[11] https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localizati
[12] https://towardsdatascience.com/review-g-rmi-winner-in-2016-coco-detection-object-detection-af3f2e

*Resnet*101). The only difference between them is that *ResNet*50 has 50 layers and *ResNet*101 has 101 layers. This difference can lead to very different outcomes in training. The smaller *Resnet*50 can be trained faster with a similar result for some tasks. On the other hand, it takes *Resnet*101 a longer training time, but has also more options to work with and is generally more precise.

3.4. Semantic vs. Instance Segmentation

Classification problems are one of the earliest tasks which deep neural networks could handle. Solving classification problems involves a network trained to categorize images. This network outputs the instances it detects in a picture. For example, the network detects a cat on an image and gives the result "*cat*" (See Fig: 3.9). This method is not able to detect multiple instances in one image. Additional, the method is not generating information on where to find the detection in the image.

The next step is to localize the instance in the image, which is called object detection.

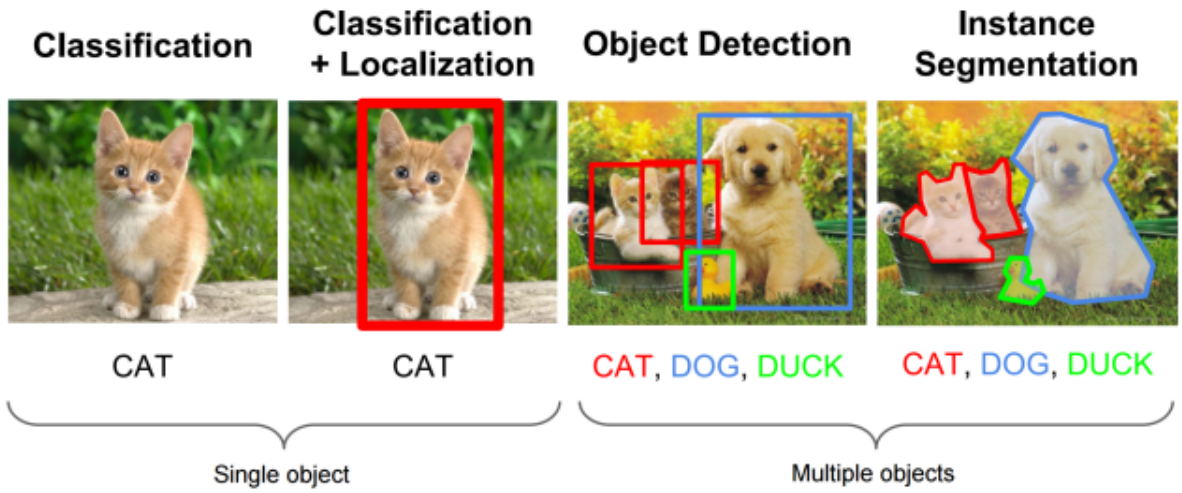

Figure 3.9.: Examples for different classification methods[13].

This strategy gives the opportunity of detecting multiple objects in one image. As shown in figure 3.9, localization is just a dare match. Therefor object detection is ideal for counting instances and for getting an idea of their overall location.

Cable detection, on the other hand, needs precise knowledge of the location of every instance, object detection is not enough. Accordingly, the next advanced method is instance segmentation. Instance-segmentation is a method whose goal is to detect every instance and its exact boundaries. As shown in figure 3.9, every animal is classified and localized in a way that the network assigns each pixel to its corresponding instance.

[13] `https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html`

3.5. Evaluating Instance Segmentation Models

The process of evaluate models for image recognition normally divides the output into four categories (*true positive*, *true negative*, $false\ positive$ and $false\ negative$). Consider instance segmentation models that produce a collection of segmentation masks in different classes. For these models, it is necessary to evaluate in a way every prediction goes into account. Also, for instance-segmentation, it is unclear what counts as TP, FN or FP. Therefore, the basic point of this chapter is to provide a sound method for evaluating instance segmentation models.

3.5.1. Intersection over Union

One solution to evaluate instance segmentation models is to count all pixels in an image that are correctly detected (Cable/Background). This number gets divided by the number of all pixels in the image. This approach is similar to the loss function and is a pretty accurate way of evaluation.

Nevertheless, this metric can be misleading when the event-size is small in comparison to the whole image. In such a case the correct detected events are small compared to the correct detected background which gives the background a higher priority.
Intersection over Union is a method to avoid this problem. The advantage of this method is that only the event is in focus and not the whole image. For calculation, all pixel representing the ground truth A (See Fig: 3.10) and all pixel detected as truth B are determent. Ultimately, IoU results by dividing the Intersection of A and B by the Union:

$$IoU = \frac{A \cap B}{A \cup B} \qquad (3.10)$$

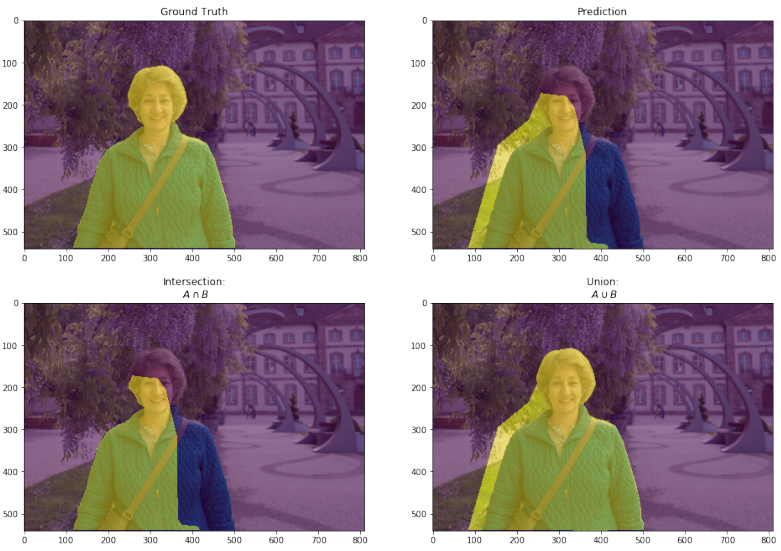

Figure 3.10.: Visual demonstration of intersection over union[14].

Intersection over union is therefore a reliable indicator for the quality of the network.

[14] https://www.jeremyjordan.me/evaluating-image-segmentation-models/

4. Proposed Method

4.1. Data Collection

4.1.1. Variety of Data

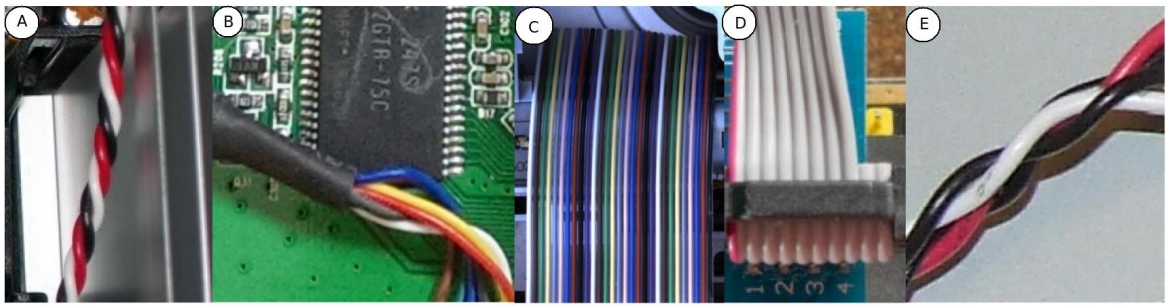

Figure 4.1.: Segments of training-images focused on a wide variety of cables.

Data collection is the most time-consuming part of training and also a necessary source to improve the network. Therefore it is important to decide which data is usable and which is not.
Before beginning to collect images, it was decided to use images which have a minimum size of **800x600 pixels**. This restriction seems to be a good standard where most cables are visible and have sharp, not blurry edges. Nevertheless, some images which had at least the minimum size were removed because some smaller cables in the image were to thin to work with.
Especially important rulers for the usability of images in cable detection are **light** and **shadows**. If shadows have sharp and defined edges, they are nearly identical to a black cable. Images were chosen to have no or at least only a few high contrast shadows to avoid confusion between shadows and cables. This strategy also leads to the fact that the later network can only detect cables on well-illuminated devices.
Even if this thesis is about cable detection, there is no sense to look at all **kinds of cable**. The main targets are isolated single wires, because of their widespread occurence and characteristic structure. Additionally, the network is trained on ribbon cables (See Fig: 4.1 C and D), cable jackets which include up to 7 single cables (See Fig: 4.1 B), twisted pair cables (See Fig: 4.1 A) and braided single cables (See Fig: 4.1 E). This restriction excludes especially but not only wires without isolation, braided metal cables, optical fibers and flat ribbon cables without noticeable structure.

4.1.2. Data Annotation

The process of making data usable for machine learning is called data annotation. Training and validation images get examined on cables which get labeled. For labeling the cables, Visual Geometry Group's (VGG) Image Annotator (VIA)[1] was used. With this

[1] `http://www.robots.ox.ac.uk/~vgg/software/via/`

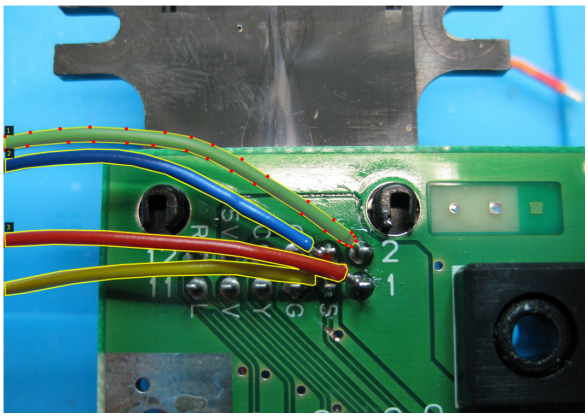

Figure 4.2.: Example annotation via VGG Image Annotator for an training image used in this work.

software, it is possible to create masks for each image by constructing polygons around the events (See Fig: 4.2). In a separate $.json-$file, that is readable by the network, VGG saves the annotation. At the end of every training step, the file is read by the network, to compare the actual place of an event with the predicted one.
The strategy obtained during annotation is to be as precise as possible. During annotation, it is not of interest how many masks cover the ground truth as long as no event gets missed.

4.2. Mask R-CNN

4.2.1. Coco-Weights

Training a new network starts typically with an empty structure where the identity function represents every neuron. These neurons get changed over time/training to get a working network. A disadvantage of this strategy is that we have to train from scratch, which can take much time.
To avoid long training, it is convenient to start from an existing network and specify it. The network recommended from the paper Mask R-CNN is a pre-trained coco weights file. Figure 4.3 shows example images and structures which build up the coco weights file. The benefit of this strategy is that the network is already able to detect structures like cats. Even if cats and cables are two different things they share common attributes like a boundary to the background and a nearly constant color over the whole structure.

[2] `http://cocodataset.org/#overview`

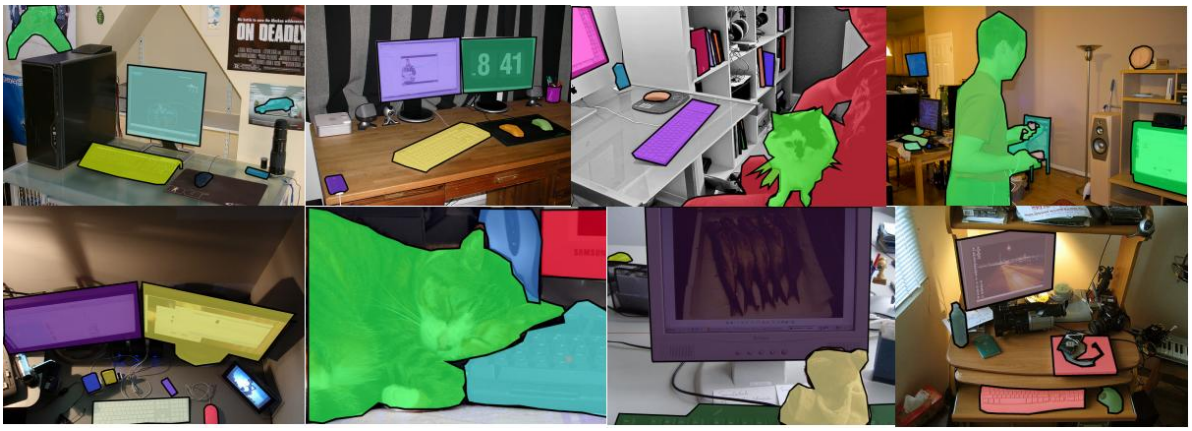

Figure 4.3.: A collection of images used to train the coco-weights-file.[2].

4.2.2. Augmentation

Augmentation is an integral part of this project to prevent overfitting and to get as much information from the images as possible. The used annotation software imagaug [19] is a library for image augmentation in machine learning experiments.
Imagug has a wide variety of image transformations. For cable detection, nearly all of these transformations create images that can occur in the later usage of the network.
This project makes usage of the following augmentations:

1. **Up-down-** and **left-right-flips** were each used 50% of the time to change the placement of the events. In the portrayed case, a flipped image is still usable because devices have no preferred viewing orientation.
   See figure 4.4 C is flipped horizontal compared to the original L.

2. Images are **croped** $\frac{1}{3}$ of the time by $0 - 10\%$ of their total size. This technique has similar benefits to flipping. The events are slightly displaced, and therefore the network gets new information
   See figure 4.4 H which is missing $\sim 5\%$ of it's width compared to it's neighbours.

3. $\frac{1}{8}$ of all images get **blurred** to make the network sensible to low quality images [20].
   See Figure 4.4 J which has the maximum blur.
   Because high-quality images should be the standard just $\sim 12,5\%$ of all images get blurred.

4. **Colour changes** which have a smaller impact on the network happens $\frac{2}{3}$ of the time to an image. Therefore a random number between 0.8 and 1.2 is multiplied to an existing color channel. The color is changed to sensitize the network to a wide variate of cable colors and changes in illumination (See Fig: 4.4).

5. In $\frac{1}{80}$ of all images, **clouds** are added to enable the network to work with low-quality images. Clouds are not a typical kind of low-quality image, but it was added to simulate a low color quality and light reflections.
   See figure 4.4 I which has maximum clouds.

Figure 4.4.: *A* -> *H*: Random augmentations of *L* generated with implemented code.
I -> *K*: Augmentation of *L* generated with implemented code.

6. Nearly every image (80%) get **translated**, **rotated** and **scaled**.
   Images get translated by $-20\%- > 20\%$ of total size which can be seen in figure 4.4 *B*.
   Scaling is comparable to zoom on a screen, the total size stays constant, and one region gets in/out focus. Images get scaled between 80% and 120% (See Fig: 4.4 *D*).
   Rotations occur between $-180°$ and $180°$ to cover all possibilities. Rotations can be seen in nearly every images except 4.4 *C*.
   These three augmentations have the goal to vary the events in the allowed area.
   The colorful background is on positions where no image is left after the above augmentation (See Item: 6). This strategy helps to sensitize the network on colorful objects that are not cables.

7. **No changes** is with 4% one of the rarest methods. In figure 4.4 *L* is an unchanged image presented. The percentage is so low because most augmentations do not lower the quality of the input. Therefore augmented images are as valuable as not augmented ones and overfitting decreases even more.

After discussing augmentations, one aspect seems to be critical: How to combine augmentation with semantic segmentation (See Sec: 3.4). The implemented code facing this hurdle works as follows:

1. Get an image and its corresponding segmentation mask coordinates.
2. Resize the image to the defined size.

3. Modify masks if image augmentation is specified.

4. Extract the bounding boxes from the masks.

Therefore every augmentation performed on an image is also done to its mask. Because masks have no color channel that could be changed, color changes build an exception of this strategy.

4.2.3. Learning Rate

LR is a vital aspect of getting good results. For finding a suitable LR, the network was trained with different constant LR and checked, which gave the best results. The resulting constant LR (C_{LR}) got varied afterward by a cycling or decreasing function to check which is the better solution.
To determine a function that enables cycling a methodical approach mentioned in [15] is a saw-tooth-function. This function decreases over time and then jumps back to the starting value.
The approach uses a repetitive cosine that converges to a saw-tooth-function due to the number of repetitions:

$$LR = a' + a \cdot cos(b \cdot x + cos(b \cdot x + cos(b \cdot x + cos(b \cdot x + ...)))) \tag{4.1}$$

Where parameters $a = C_{LR}$ and b are determent by testing constant LR's and setting $a' > a$ to avoid $LR <= 0$.
The most promising saw-tooth function is:

$$a' + a \cdot cos(b \cdot x + cos(b \cdot x + cos(b \cdot x + cos(b \cdot x)))) \tag{4.2}$$

Where a and b had to be determent for each optimizer separately, and x corresponds to the epoch. For determining a decreasing function, the constant LR C_{LR} is a set midpoint. After at least 50 epochs this midpoint should become the actual LR. This boundary condition leaves the network enough time to find global minimums in the beginning and progress to the bottom during the later epochs.
The used function is a repeated if-statement which sets a new LR after each 10 epochs:

$$LR(Epoch) = \begin{cases} d/10, & \text{IF Epoch} > 10 \text{ AND Epoch} \leq 20 \\ d/50, & \text{IF Epoch} > 20 \text{ AND Epoch} \leq 30 \\ d/100, & \text{IF Epoch} > 30 \text{ AND Epoch} \leq 40 \\ d/500, & \text{IF Epoch} > 40 \text{ AND Epoch} \leq 50 \\ d/1000, & \text{IF Epoch} > 50 \end{cases} \tag{4.3}$$

In equation 4.3 $d = 500 \cdot C_{LR}$, to generate a high LR in the beginning.

5. Experimental Evaluation

The final training took $\sim$ 55 hours (See Fig:5.1 Right) for 650 epochs with 200 steps per epoch on an NVIDIA GeForce GTX TITAN X 12 GB graphics card. 2 images were processed simultaneously on the graphics card. For the first 86 epochs, training only included network heads, followed by the remaining 101 layers. Accordingly ResNet 101 is the used backbone. ADAM is the used optimizer while training with a learning-rate strategy discussed earlier. Appendix section *Step* 650 contains 49 images of electronic devices with the associated network predictions.
Figure 5.1 left shows that the validation loss is not lowest at step 650 with a value of 2.615. However, step 650 is a good approximation of the last 20 steps. One of the lowest validation losses is at step 648, with a value of 2.045. The resulting network of step 648 was tested to get an understanding of the underlying structure.

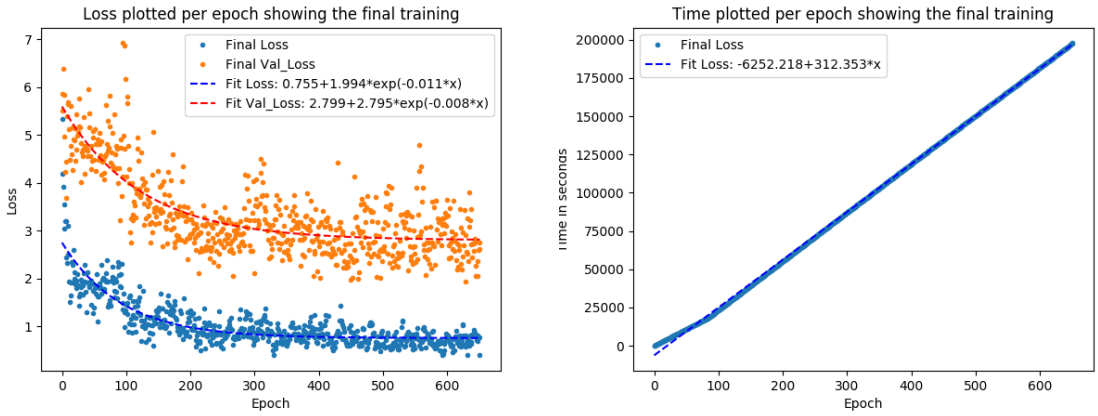

Figure 5.1.: Final losses, data generated via TensorBoard[1]. **Left:** Loss per epoch. **Right:** time in seconds per epoch

5.1. Intersection over Union

The IoU is calculated in the way introduced in chapter Background/Theorie. Because multiple images were detected, Intersection and Union are calculated separately for every image. The resulting Intersections and Unions are summed separately. Afterward, the

[1] `https://www.tensorflow.org/tensorboard/r2/scalars_and_keras`

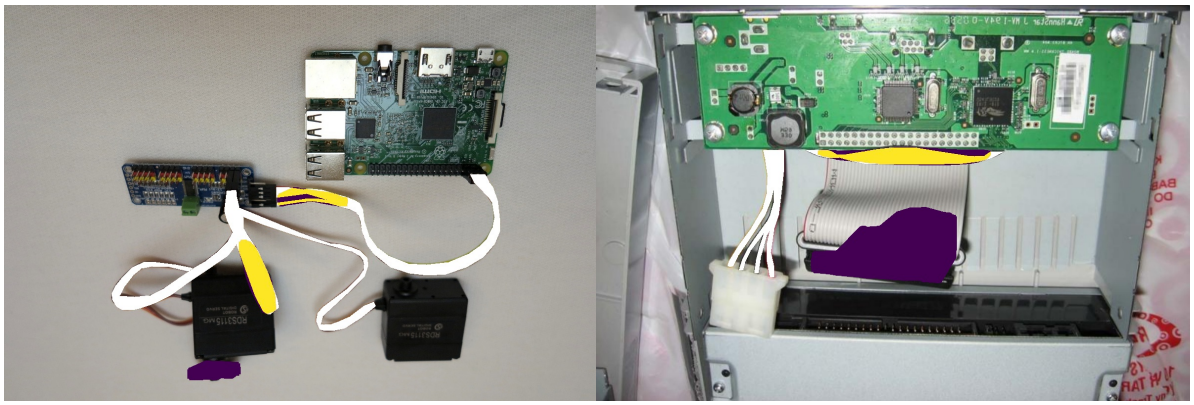

Figure 5.2.: Intersection over Union on common setups. White: Ground Truth, Violet: Detection, Yellow: Intersection

sums are divided by one another to calculate the IoU. This method is used to get a statistic value where all misinterpretations are equally weighted.
Another approach commonly used to calculate the IoU is for each image separately and averaged. The second approach would result in better IoU. This approach would handle an image with no detection the same way it would handle an image with only wrong detection with a resulting $IoU = 0$. Accordingly, the first approach misinterpretations have a more significant impact due to the summed Union.
The resulting IoU's are:

$$IoU(650) = 0.053211494700403913$$

$$IoU(648) = 0.051036857157608556$$

The formulated goal is to detect cables in order to remove them in a way that the safe removal of all components is ensured. For a safe removal it is not necessary to detect entire cables as long as each cable has a detected section . To get an idea of how usable the network is, all correctly recognized events are counted and compared to misrecognized structures and unrecognized cables.
Observing the 49 images in appendix *Step* 650 which contains in total 159 cables, the network detected 43 of them. On the other hand, the network misinterpreted 24 structures as cables. For example shadows (See Fig:A.17) and light reflections (See Fig: A.21) are a source of misinterpretation, but also structures which are in a similar pattern as cables (See Fig: A.22).
On 6 images of 49, the network found all cables and no misinterpretations occurred. These images make up 12.245% completely correct detected images.
In total every $\sim 5^{th}$ cable was recognized by the network. Additionally, 17 images contained at least one misinterpretation, which is roughly every third image.

6. Discussion

6.1. Strategy

6.1.1. ResNet

ResNet is the recommended backbone for this task. As mentioned earlier, ResNet50 and ResNet101 are the possible backbones for Mask R-CNN and got tested on the presented problem.

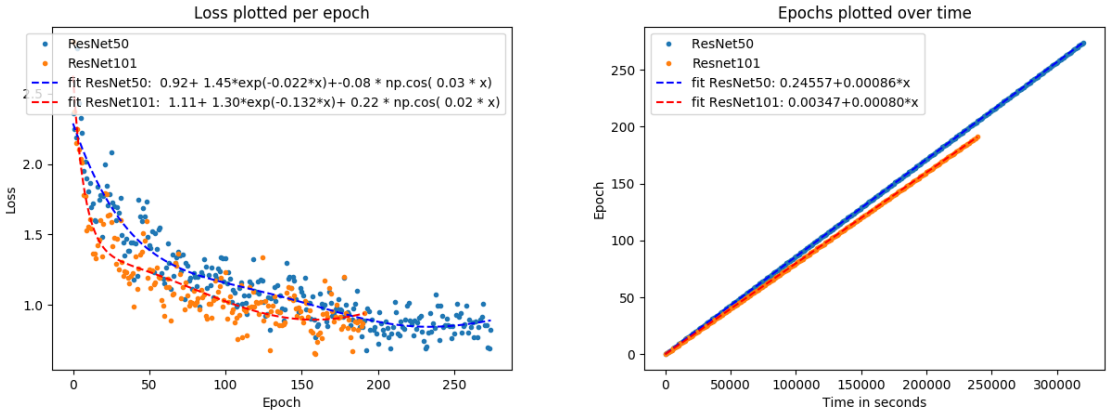

Figure 6.1.: Left: The loss of ResNet50 and ResNet101 plotted over epochs.
Right: Epochs of ResNet50 and ResNet101 plotted over time.

By comparing the training loss, it is easy to see that both backbones create similar results (See Fig: 6.1 A). ResNet101 loss is decreasing fast in the beginning but at epoch $\sim$ 150 both losses approach each other.
The used fit is an exponential function overlaid by a cosine:

$$\text{Loss}(\text{Epoch})_{\text{Fit}} = a + b \cdot exp(-c \cdot \text{Epoch}) + e \cdot cos(d \cdot \text{Epoch}) \tag{6.1}$$

Even the time used for training is comparable (See Fig: 6.1 B), while ResNet50 needed $\sim$ 19.5 min per epoch ResNet101 needed $\sim$ 21 min per epoch.
By looking just at figure 6.1, ResNet50 is the better backbone. ResNet50 is faster and delivers just slightly worse results at the beginning of the training.
The real deficits of ResNet50 is the bad performance on validation data (See Fig: 6.2). In the figure, ResNet50 is getting constantly worse on validation data. This behavior is an indicator of overfitting. Nevertheless overfitting is not very likely because the bigger network ResNet 101 constantly improves without a sign of overfitting. Another explanation is that ResNet50 can only save some specific features while missing the overlaying structure.

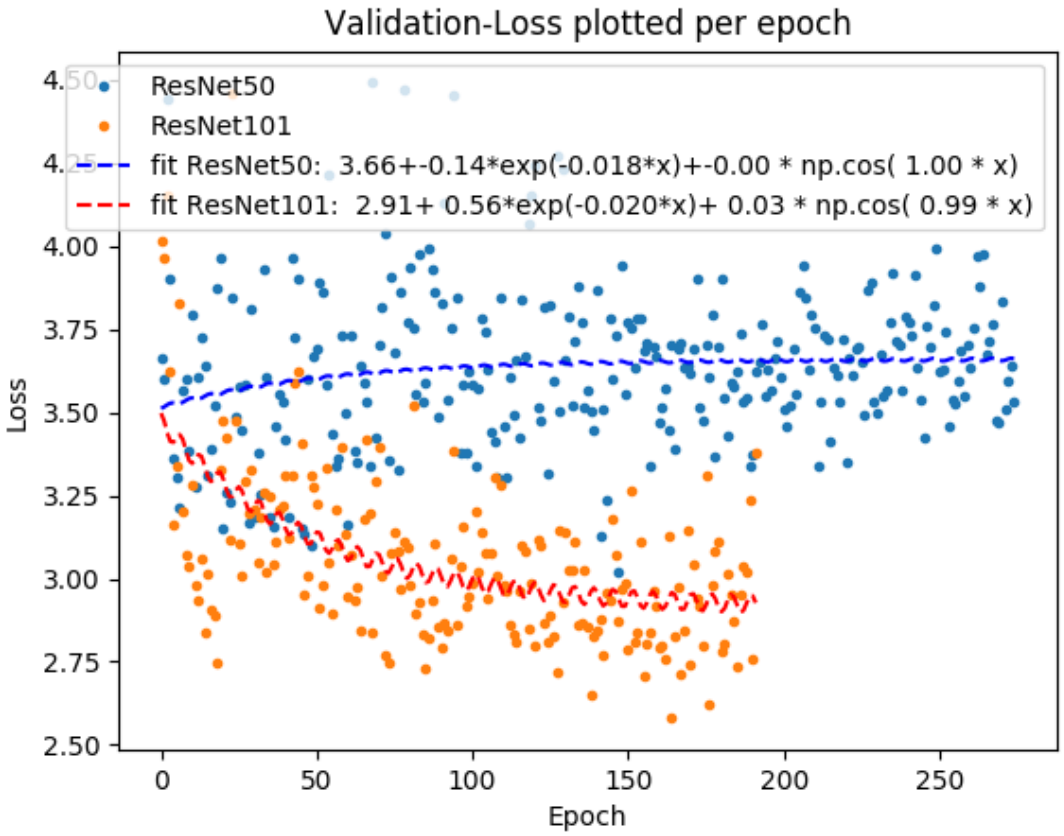

Figure 6.2.: Validaton Loss of ResNet50 and ResNet101 compared.

Combining these results, ResNet101 is the better choice even if it takes more time to train.

6.1.2. Learning Rate

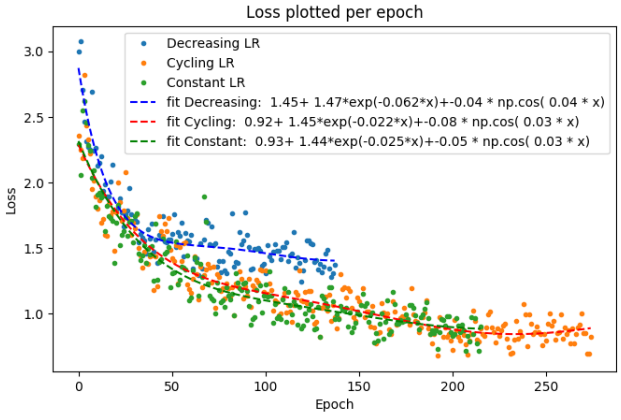

Figure 6.3.: Loss for the discussed LR-strategies over the first $\sim$ 150 epochs.

Comparing the resulting loss of learning rate approaches (See Fig:6.3) the constant and the cycling LR are similarly good. Just a look at the fit shows that cycling is slightly better because the exponential function decreases faster. The decreasing LR made a relative lousy loss, which results from the fast decrease in equation 4.3. The steep descend lets the LR jump in the beginning and the network cannot change to a usable minimum.

Figure 6.4.: Results of an short training using both optimizer.

Because the final network contains more data and extended training, cycling LR seems to be the best solution.
It can also be concluded that the LR strategy has only a small impact on the quality of the presented network. The decreasing LR made a lousy impression mostly because the parameters were not well set. Other works suggest a decreasing LR as a soiled alternative which generally improves the network. Therefore is the presented solution not representative of other works.

6.1.3. Optimizer

For **ADAM** the constants in equation 4.2 were chosen to be $a = 0.000055$, $b = 0.26$ and $a' = 0.0000551$. These are slightly different constants then for **SGD** were $a = 0.00011$, $b = 0.20$ and $a' = 0.00011$. Both optimizers worked better with different LR's. ADAM is, for example, not able to work with such high LR's. High LR's cause the network quality to decrees due to growing losses. On the other hand, SGD not makes much progress over epoch with low LR's. A low LR additionally minimize the benefit of cycling LR because only the high LR make progress during training.
Comparing both optimizers (See Fig: 6.4), ADAM makes a good impression. It falls faster than SGD, and both optimizers stop decreasing noticeably at roughly the same epoch. Therefore ADAM is used in the final training with the same parameters mentioned above. The resulting loss function shows the differences in the optimizer. In this specific task, ADAM is the better option. Using ADAM, the loss decreases much faster and produces a network with a lower loss. Therefore has the usage of a suitable optimizer a positive impact on the later results.

6.2. Cable Structure

The structure of cables is due to its form and texture fundamentally different from most other objects detected with Mask R-CNN.
In related work it is discussed that normally an edge map is used to determine the events. For many of these works, ground truth could be detected by finding big structures and then interpret the inner texture.
In figure 6.5 edge detection was used on images that had to be detected. Some differences are visible comparing the edge detection used on cable images and those used in related works (See Fig:2.2). While edge detection fits needle, the events in related works, cables are often just detected part-wise. Additional have many other structures surroundings similar to those of cables (See Fig: 6.5 C+D).
In related works, the only significant edges are those of the events. This overflow of edges in the edge-map makes a clear assignment of cables difficult.

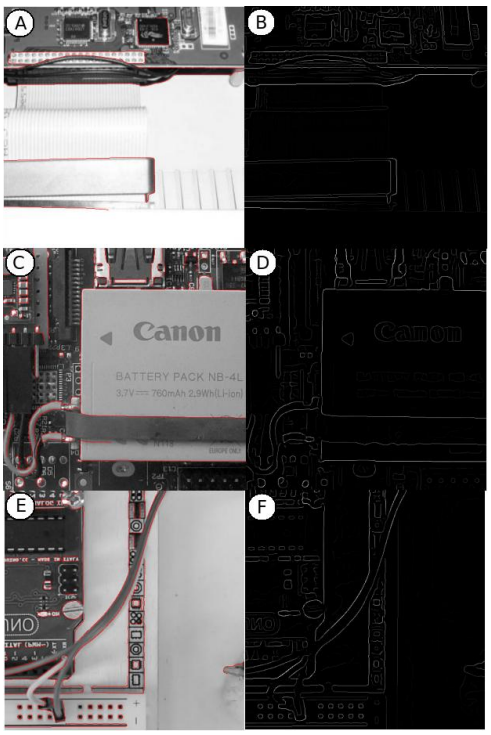

Figure 6.5.: Edge detection processes on images of devices. (**A+B:** A.6 left ,**C+D:** A.15 right, **E+F:** A.2 left)

6.3. Loss

In figure 5.1 the validation loss in comparison to the training loss is shown. One vital aspect is that the validation loss is much worse than the training loss. Interestingly, both loss functions have nearly the same slope and just another loss in the beginning. Overfitting can cause validation loss to become much worse than the training loss. As discussed earlier can overfitting detected when the validation loss starts to grow at a particular step, the training loss decreases drastically. Because the validation loss is continuously decreasing and the still high training loss has a constant curve, overfitting is less likely.
Another project handling overfitting brings clarity. The work for automated characterization of arctic ice-wedges 6.7 handles overfitting and also uses Mask R-CNN. The loss presented in the paper began to overfit after epoch 8. Therefore the resulting network of epoch 8 was used for later detections. The overfitting can be seen precisely at step 8 at which the validation loss curve starts to grow drastically while the training loss is still decreasing.
For detecting the point of drastic growth, a network with the same parameters as earlier and only a different step size of 20 was created. This network demonstrates the tendencies of the presented network (See Fig: 6.6) in early epochs. Additionally, all 86 validation

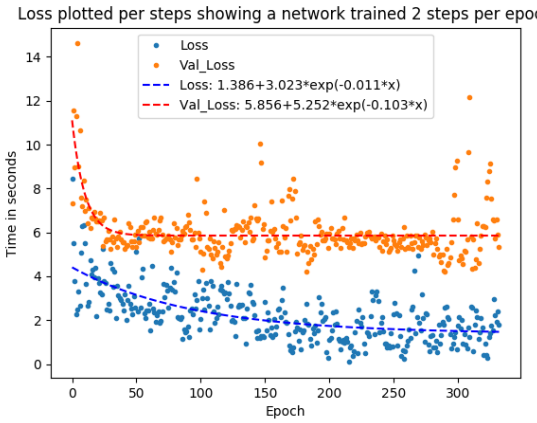

Figure 6.6.: Loss curve for a networks trained 20 steps per epoch for detecting overfitting.

images are used each epoch to check the resulting network. This strategy allows a constant baseline to check the network on.
The resulting loss function is very different from the one which shows overfitting in earlier works (See Fig: 6.7). The presented network has huge fluctuations between some epochs, but the overall curve is decreasing just as the final network. Because training involved 305 images, not every training image is used until step 15. Therefore overfitting is impossible before step 15 because not every image has been processed so far. However, at step 15 the validation loss is already much worse than the training loss. Combining all this information overfitting is not the reason for the lousy validation loss.

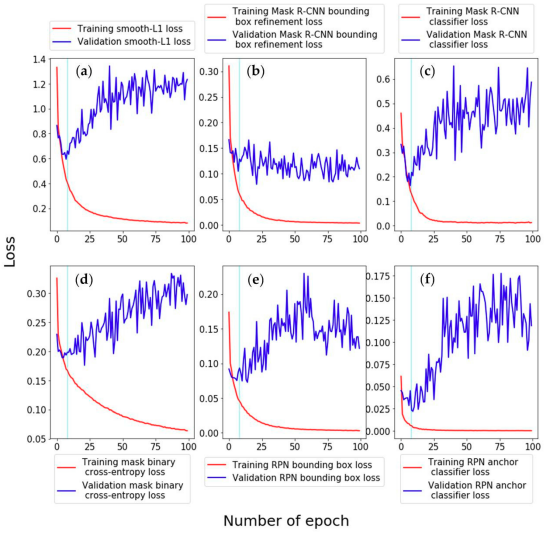

Figure 6.7.: Loss curve form a Mask R-CNN network designed to detect ice-wedges[1].

It is also possible that the validation data is treated differently from the training data somewhere in the code, which causes this strange behavior. Such a failure is difficult to find in the code as long there is no concrete plan what has gone wrong.
Another explanation could be that training did not include enough images for a considerable variety of cables. The lack of data could lead while validation to color and shape combinations new to the network.
Additionally, a hidden, systematic error could cause this behavior. Even if the images were chosen randomly as training or validation image, there could still be some hidden differences between both datasets. For example, cables in images for validation cover an allover higher area. Cables in validation images cover in average $\sim$ 4.989% of the entire image and training images

[1] `https://www.mdpi.com/2072-4292/10/9/1487/htm`

just $\sim 3.955\%$. This difference would result in worse validation-loss regarding the overall low detection probability.

6.4. Intersection over Union

The IoU for both networks is roughly the same with a score of ~ 0.05. Normally a score of > 0.5 is considered as a good prediction[2]. Therefore the resulting network can not be considered as a working network. The resulting predictions are most of the time too small or not overlapping the ground truth.
The first object to discuss is the relatively small difference in IoU between network 650 and network 648. Just considering the validation loss, the network 648 has better values. Inspecting only the IoU, both networks are equally good. Therefore the loss function doesn't seem to be a fitting way to detect the quality of a network.
As discussed earlier, IoU is the better way to determine the quality of a network designed for detecting small objects. While the loss function is taking every pixel into account, IoU dismisses the background. This disadvantage has an impact on the precision of the network. The predictions of network 650 cover an average area of $\sim 2.678\%$ compared to network 648, which covers an average area of $\sim 0.473\%$. Network 648 covers a much smaller area than network 650 with a similar IoU. This behavior partly results from detection with a wider area than the ground truth. These detections often cover gaps between two or more cables which are too small for separating them from cables. Therefore one improvement between both losses emerges from the smaller predictions network 648 makes. Cables have an overall small area which doesn't have a significant impact on the loss function. Even an image without predictions could have a satisfying loss as long the ground truth is small enough. This behavior could lead the network to detect only the cable it is absolutely sure about and to shrink the prediction as far as possible.

7. Conclusion

7.1. Detection

The network is barley able to detect cables. Nevertheless, it is more confident when cables overlap (See Fig: A.19 left + A.26 right) and when the surface reflects light (See Fig: A.13 right + A.5). This behavior seems to be understandable due to the networks approach to first searching for edges and afterward interpreting the color spectrum. Due to the color spectrum, overlapping cables and light reflections are recognizable compared to single one

[2] `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`

colored cables. The downside of this behavior is that the network often detects structures which are no cables. This misinterpretations occur were light is reflected from the casing (See Fig: A.12 left + A.21 right) or parallel lines are carved into parts (See Fig:A.26 left + A.22 right). Additionally, the network only detects specific overlaps and light reflections. Some high-resolution images with many overlapping cables and with light reflections are not detected (See Fig: A.12 left + A.15 left). The networks seem to have problems with cable colors and cable structures it has not seen before.
Detecting single cables doesn't seem to follow some apparent reasons. If the network detects a single cable, it often detects the entire cable (See Fig: A.17 left + A.1 right). In the few images where single cables are detected the background is mostly monochrome and the cable is not in direct light. Additionally the color of the cable has no noticeable impact on the prediction. The network detected all kinds of color as long as the background is in a noticeable different color.
Detecting ribbon cables depends on the light it reflects and the shadow it produces. The more the structure of parallel cables appears, the higher the chance is, that the ribbon cable is detected (See Fig: A.6). Therefore the network only detects regions that are well illuminated. The most problematic ribbon cables are white and flat ones, which are missing inner structure (See Fig: A.20 left + A.4 left). These cables are sometimes indistinguishable from white surfaces like labels or warning signs. Therefore flat ribbon-cables are generally not counted as cables in this thesis. Anyway the network is producing misinterpretations due to the confusion between ribbon cable, single cable, and white background. Earlier attempts to include flat ribbon cables failed due to these problems. Therefor, the huge detection's produced on flat ribbon-cables are no correct predictions and damage the quality of the network (See Fig: A.6 left).
Just counting the numbers of detected cables is also not promising. The network detected only one cable out of four. This result is far from usable in a disassemble environment. Especially while noticing that misinterpretations occurred 24 times. In conclusion, every third detection is a misinterpretation and would disturb disassembling devices.

7.2. Data

The dataset is the most important subject for a well working network. It is essential to collect enough data and to annotate them as needed. Most other projects using Mask R-CNN include a dataset of more than 5000 images. These images are often especially taken to fit the later usage of the network. The presented project roughly includes 400 images. Therefore, it can be seen that this dataset does not contain enough information for a complex structure like cables.
An additional disadvantage is that most images are obtained randomly from the internet. This broad field of sources results in many different color spectrums, camera angles, qualities, cable shapes, and lineage country. Therefore the dataset also includes shapes and colors which the network will never face in the later usage.
Cable detection in old devices is a tough task, for instance, segmentation. Primarily due to the many edges in devices. Each separate part can form edges that are similar to cables. Especially parts from similar materiel (See Fig: A.12) are hard to separate from

a cable.
There are not only many edges around cables but also sometimes inside them. For example, when ribbon cables run below other parts or cables. The shadow can result in an edge (See Fig: A.21 left) which forms a detection. The Resulting edge can include a random area, depending on the threshold of the edge detection. Additionally, the many structures inside a bent ribbon cable can form edges which have an impact on the prediction. In (Fig: A.9 right), a ribbon cable has a knot at one end. The network detected the part without the knot, but not the part with a knot.
In summary, the used dataset is much too small and not created in any disassembly environment were the quality of the images is constant.

7.3. Loss-Function

The loss function used in Mask R-CNN is not ideal for detecting small area objects. As explained earlier, the network is more eager to predict the background right, because it has a more significant impact on the loss due to its area. The detection of cables is secondary because they only cover an area smaller than $\sim 5\%$. Additionally, when the prediction is vaster than the ground truth, the benefits of detecting the truth are decreasing.
Consider detecting cables with a small area and which are parallel to each other. These structures have often spaces between them which cover the same area as the cables itself. Therefor detecting parallel cables, including spaces between them, can be a disadvantage for the loss function.
A better approach would be to use IoU, which only focuses on the prediction and the ground truth. This evaluation method would highlight the cables, and ignore the background which has no impact on the IoU. If the same setting with parallel cables get evaluated by IoU, the same problems occur. The benefit of IoU is that the prediction would give a relatively bad outcome compared to the outcome of the loss function. Therefor the network has a bigger incentive to overcome this problem

7.4. Usability

The network produces many misinterpretations and is not able to detect a sufficient number of cables. Therefore the network is not usable in the current state. Nevertheless, the network detects cables and is confident if detections occur. Regarding the dataset, the network is trained on, the results are satisfying. Even this small, not specified network was able to detect cables in random settings. Suggesting the same network is trained on one kind of device in a known setting to achieve cable detection. Such a standardized setting has the potential to improve cable detection far enough to make it usable in disassembly processes.

8. Future Work

8.1. Collect Data

The most significant disadvantage while facing this problem was the small amount of data. For training, a sufficient network one usually has to collect > 5000 images.
To achieve that considerable data size, searching through the internet can not generate enough data and annotating all these images is a dulling task.
Another approach facing this shortcut could become synthetic data. This method was used in paper [7] and delivered reliable solutions. The efficient image generation proposed in the paper helped to improve the network to become usable.
Even with synthetically generated data, cable detection can become tough due to the wide variety. This wide variety would also be a downside of synthetic data because the data has to be prepared for all eventual settings. These preparations would include hundreds of different cable colors and possible bindings. Also, the synthetic data is not as good as actual images. The presented synthetic data had problems to adjust the shadow right and to place the events naturally. Even this difference between real data and synthetic data could have a significant impact on actual cable detection.

8.2. Detection of Cable-Types

One of the next steps is to extend the network to differentiate cable types. As mentioned above, the network is specified to single cables with almost only colored isolations. Therefore many kinds of cables will not be detected.
The benefit of separating cable types is the freedom to decide what happens to the cables after detection. For example, regular cables contain metals like copper and aluminum, which is not the case with optical fibers. Respectively both have to be recycled separately.
An additional benefit of separation between multiple cable types is that it could improve the accuracy of the network. If the network is trained on different cables, it can calculate a strategy for each type of cable. The strategies would include different standards for ribbon cables and single cables. Structures similar to ribbon cables are often a source of misinterpretations. However, if the network is not looking for ribbon structures in every area, it would reduce Misinterpretations of these structures.

8.3. Depth Sensor

As explained earlier, the aim is to remove cables from old devices. To accomplish this goal, adding a depth-sensor to the setup could be necessary. There are two possible solutions on how a sensor could improve the setup.
First, the depth sensor can be used to determine the distance to the detected event. Therefore the network provides the 2D position of the object to the depth sensor. The

senor is than placed above the determined area to check the height. This information can then be used to cut the cable without damaging other parts. Without a depth sensor, this could also be manageable with a pressure sensor.
Nevertheless, a depth sensor is the more straightforward solution due to the vast area it can oversee.
Secondly, the depth sensor can be used to generate data. With a suitable depth sensor, the generated data can be an additional parameter for the images. Because cables have an ample color space, the position information determined trough edge detection is the most valuable information. Adding a position parameter would, therefore, improve the network a lot.

8.4. Personalize the Network

One major problem while training was the wide variety of camera angles, camera distances, image qualities, cable types, and countries of origin. These problems can not be solved in general because it would take a massive amount of data and an even deeper network to safe all possibilities. Therefore to make the network viable in the recycling sector, it is essential to specify it on the used equipment and the variety of e-waste. A supervised disassembly could do this specification. The idea is to retrain the network with actual data from the disassembly environment. Therefore a human has to check the predictions of the network and correct it if it is mistaken. This supervision has to be done even without data collection, to ensure that everything works fine and cables do not get overlooked. The idea is therefor that an inspector checks every prediction and annotates missed cables. This strategy could generate a large dataset with real-life images in a short amount of time. When enough data is collected or after a specific amount of time, retraining the network starts. For training, the already existing network can be used as a starting point to save time.
This strategy should generate a network with only small errors which could make the inspector unnecessary. Consequently, new training and supervision only have to be done when new equipment or new kind of devices are processed

A. Appendix

All images shown in this chapter are obtained from Google images [1]. The images presented here are not my own work. I checked the images for cables with the discussed network. Therefore I added the detection masks to the images.
All images were chosen to fit the criteria discussed earlier and to check the quality of the network. Also, it was necessary to get images that have not been used earlier. Therefore all images have been checked via the image compression software pHash[2].

A.1. Step 650

[1] https://images.google.com/imghp?hl=en&gl=br
[2] http://www.phash.org/

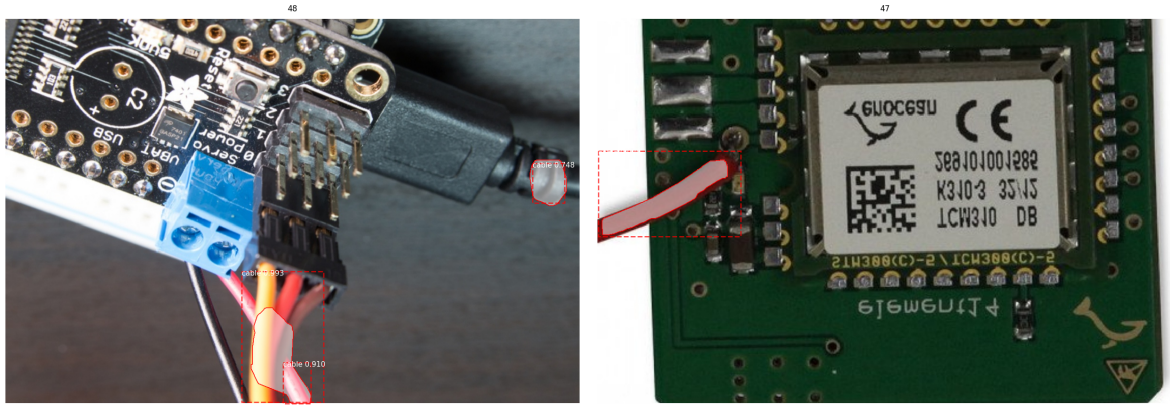

Figure A.1.: Cable detection in common setups.

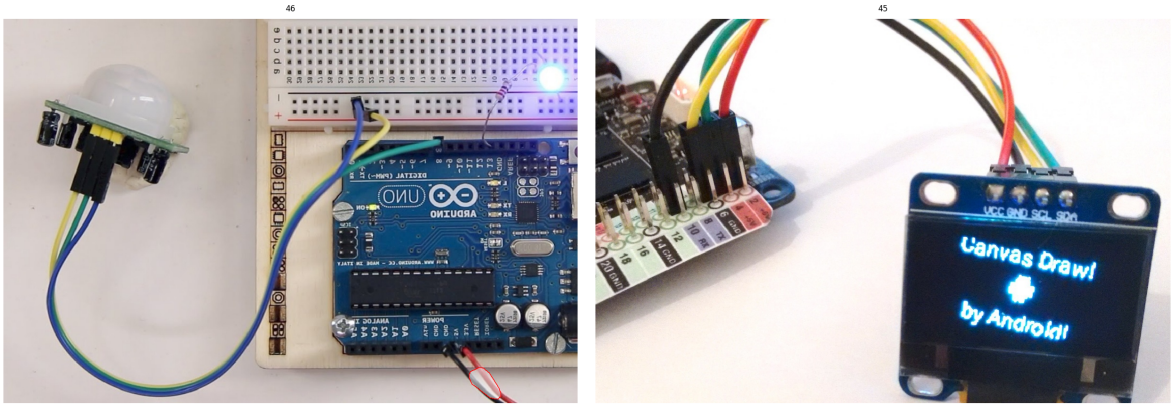

Figure A.2.: Cable detection in common setups.

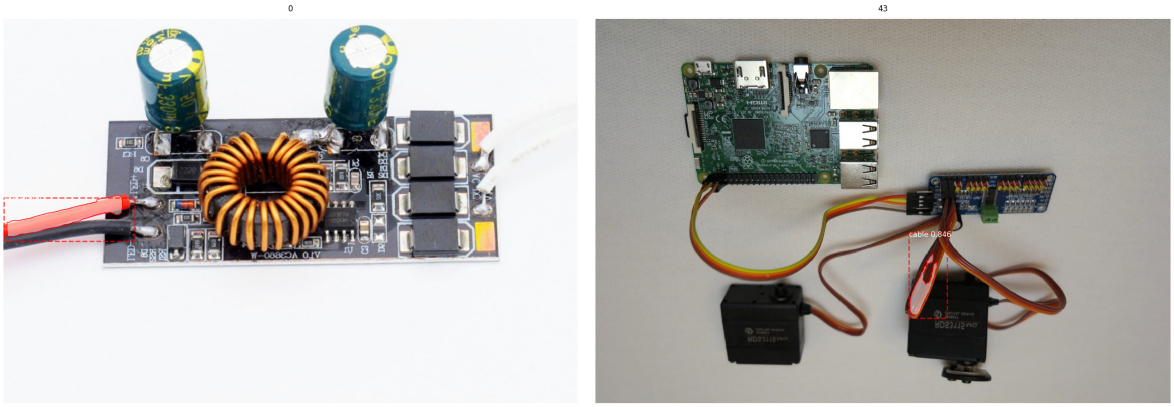

Figure A.3.: Cable detection in common setups.

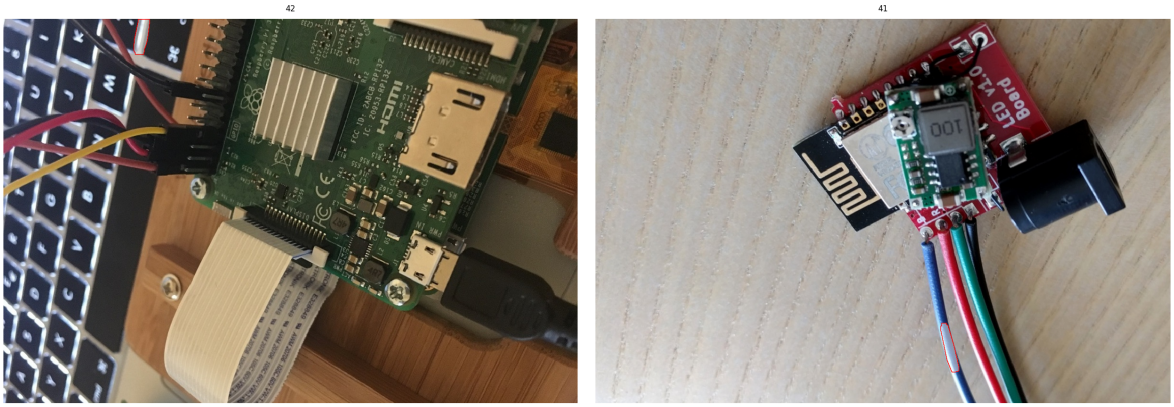

Figure A.4.: Cable detection in common setups.

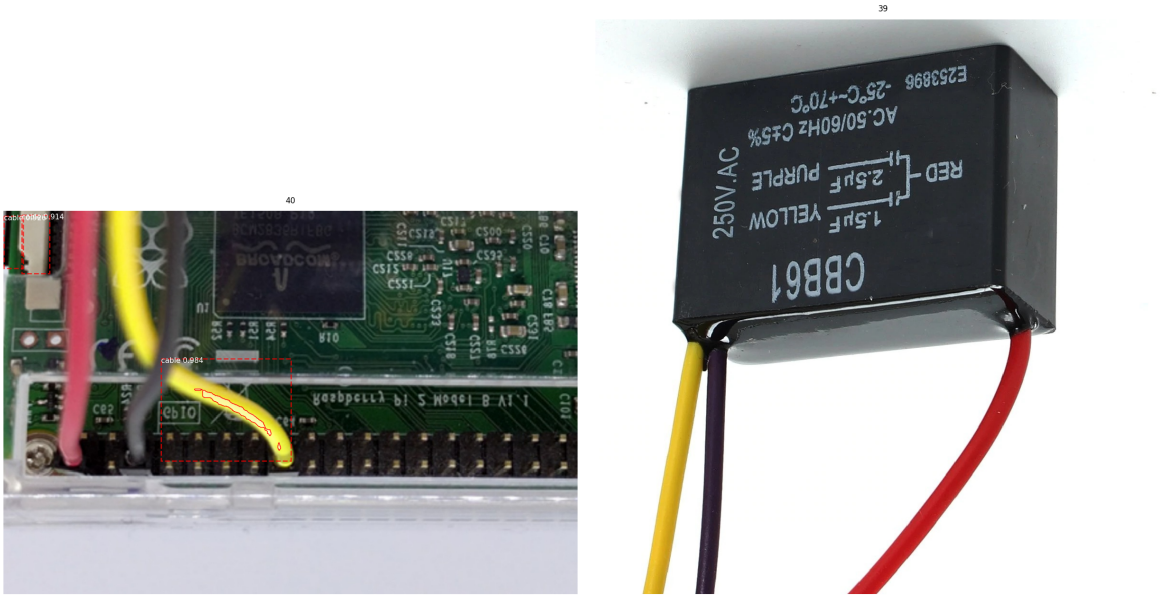

Figure A.5.: Cable detection in common setups.

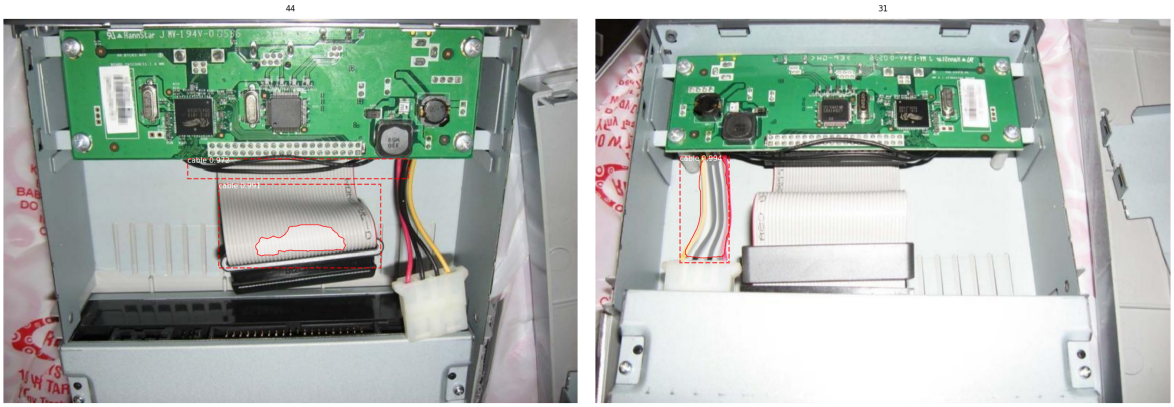

Figure A.6.: Cable detection in common setups.

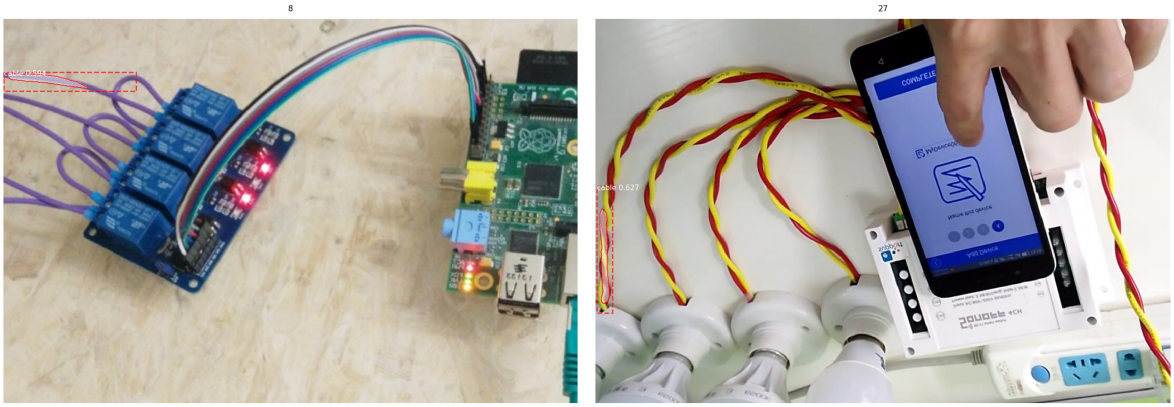

Figure A.7.: Cable detection in common setups.

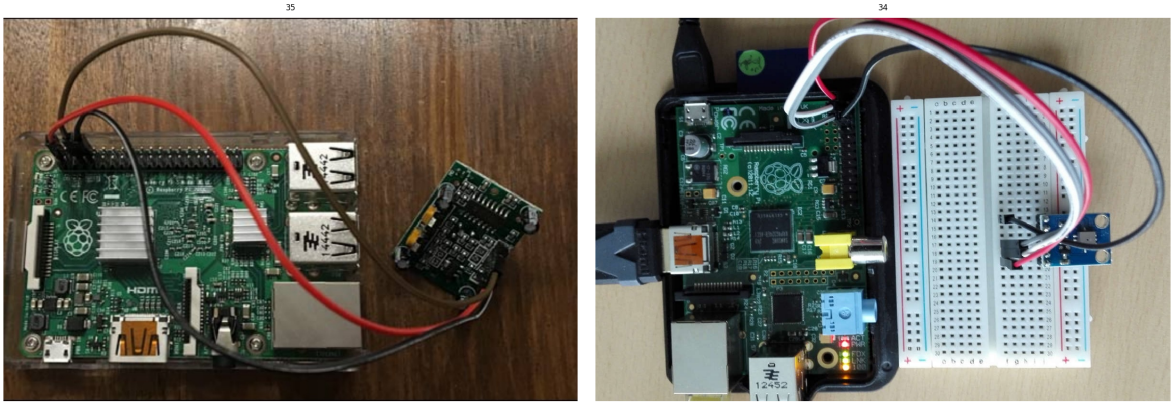

Figure A.8.: Cable detection in common setups.

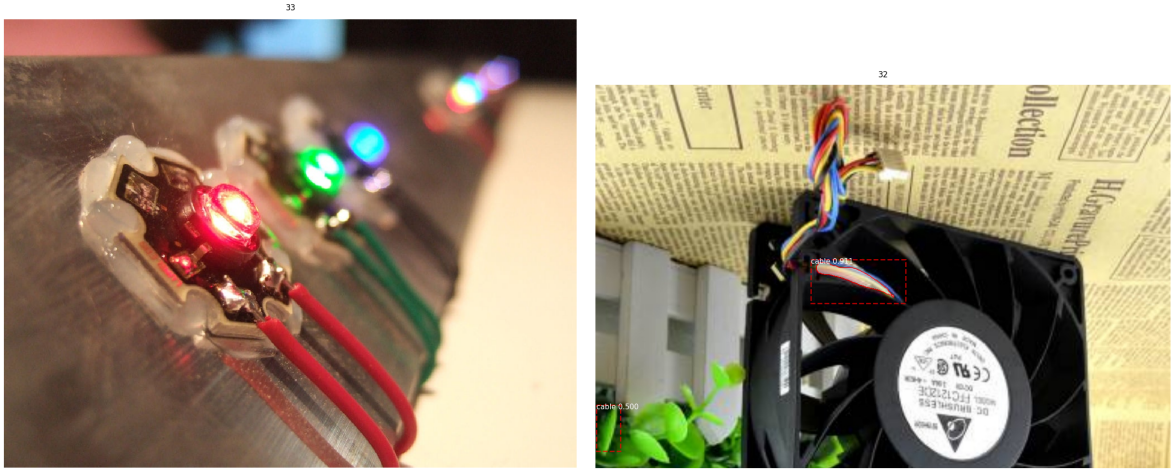

Figure A.9.: Cable detection in common setups.

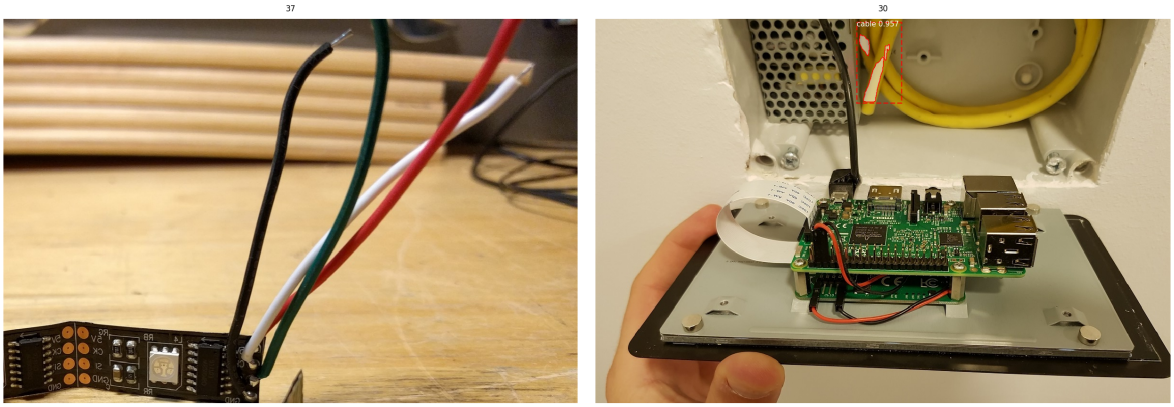

Figure A.10.: Cable detection in common setups.

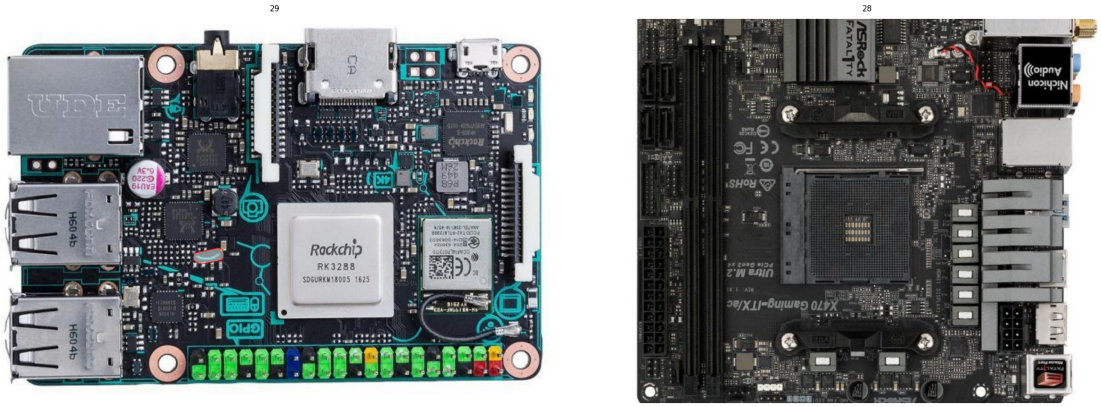

Figure A.11.: Cable detection in common setups.

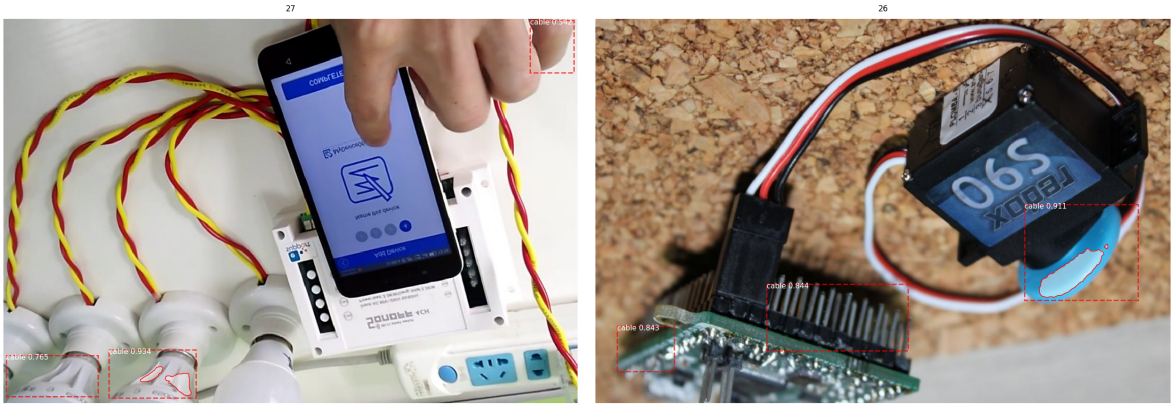

Figure A.12.: Cable detection in common setups.

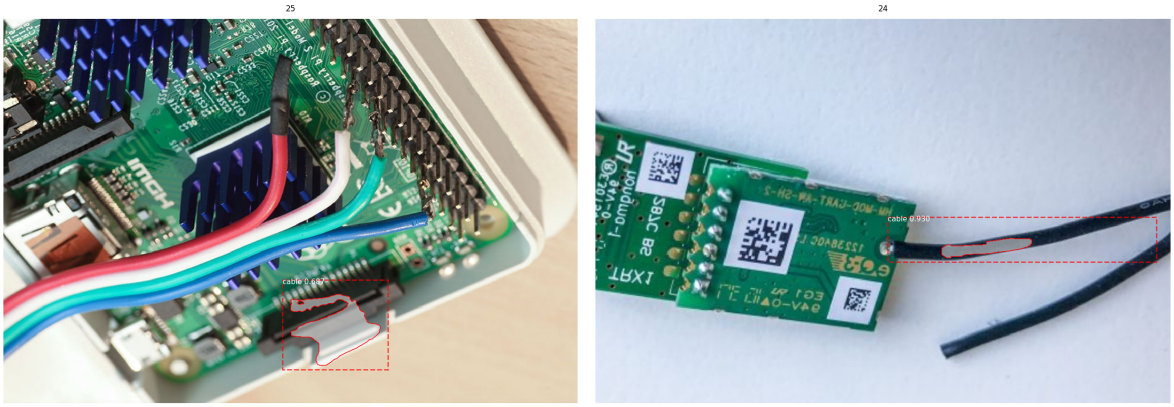

Figure A.13.: Cable detection in common setups.

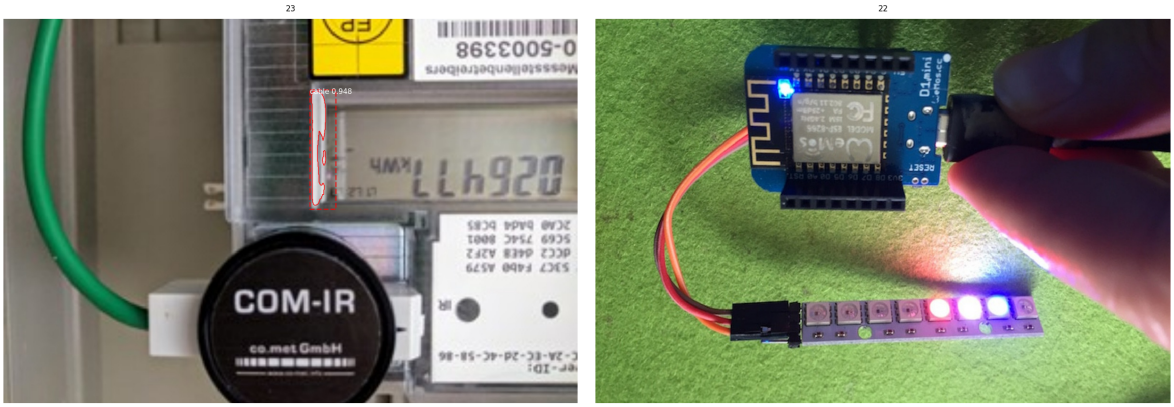

Figure A.14.: Cable detection in common setups.

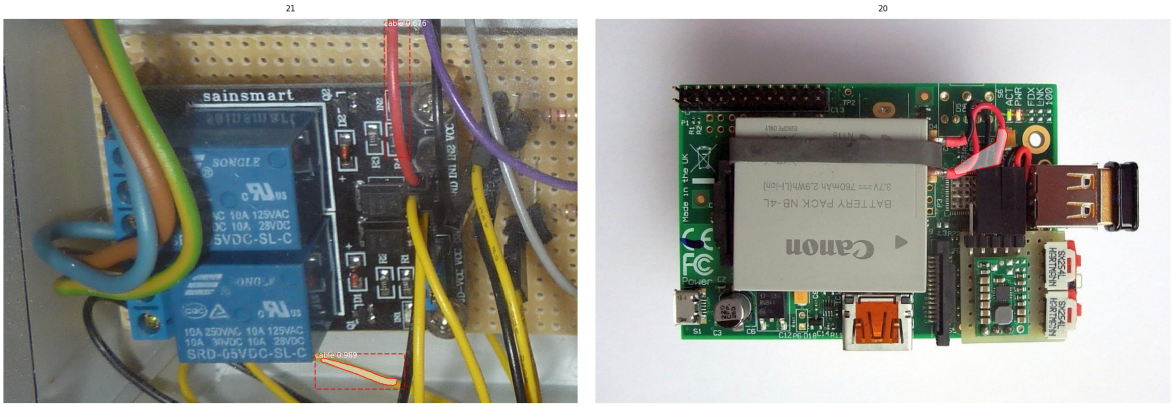

Figure A.15.: Cable detection in common setups.

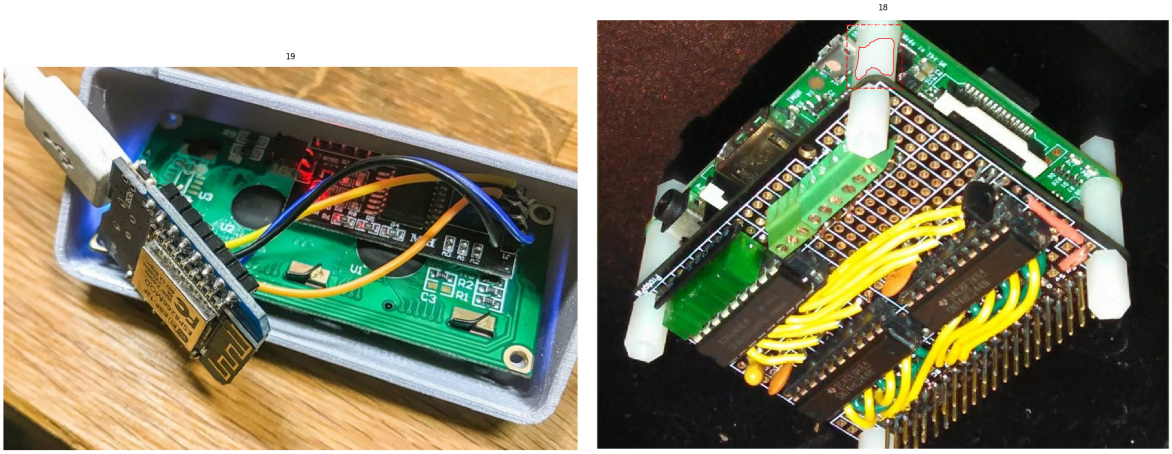

Figure A.16.: Cable detection in common setups.

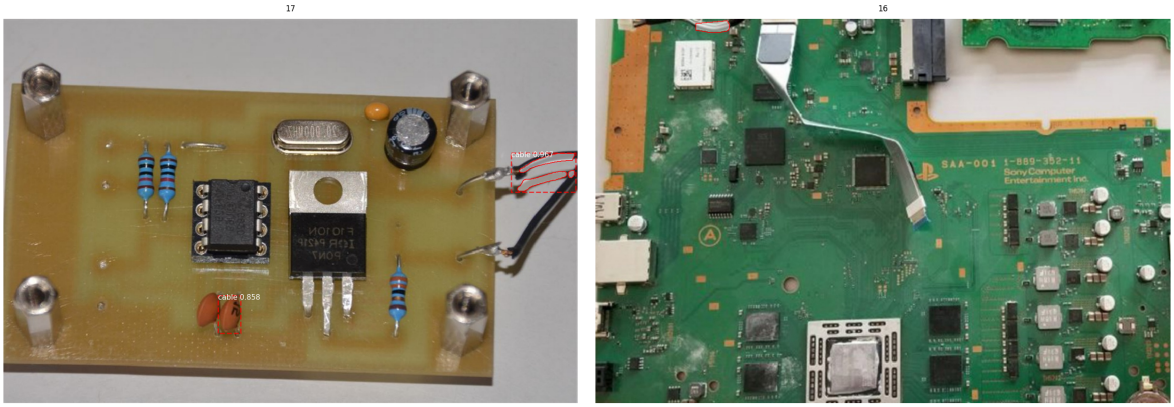

Figure A.17.: Cable detection in common setups.

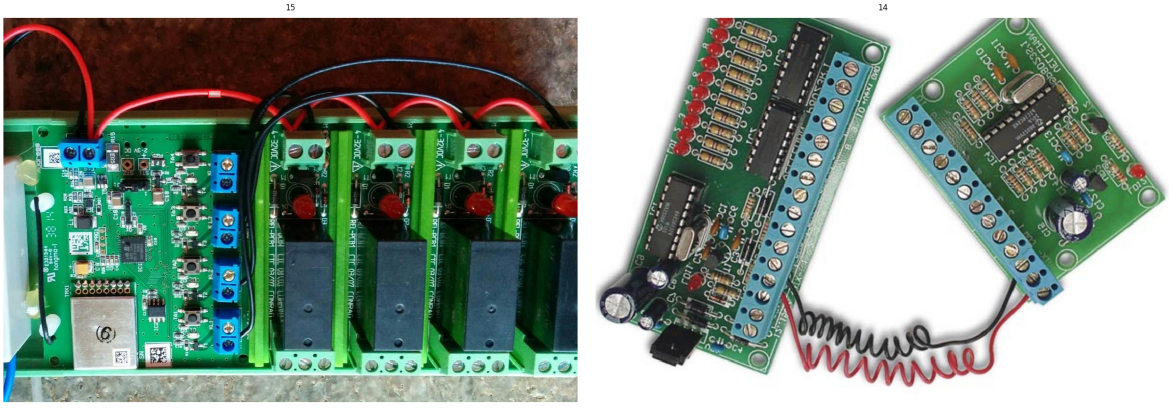

Figure A.18.: Cable detection in common setups.

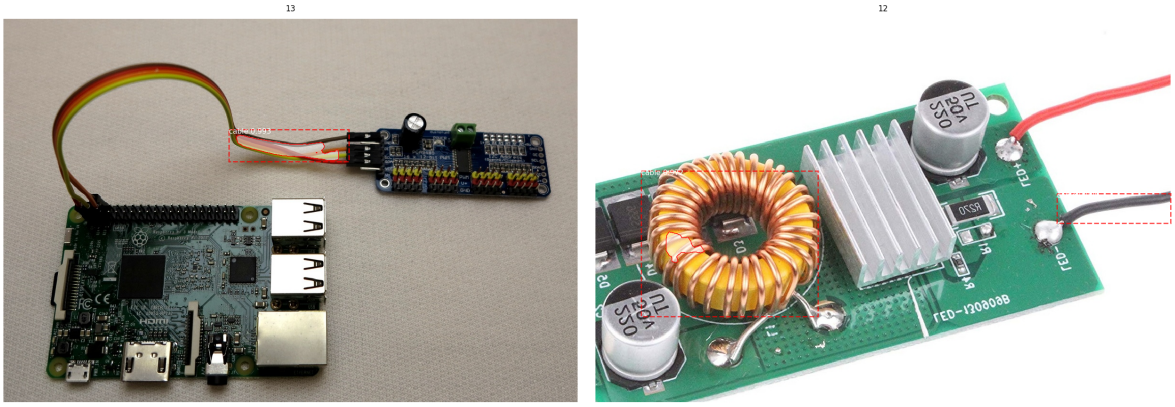

Figure A.19.: Cable detection in common setups.

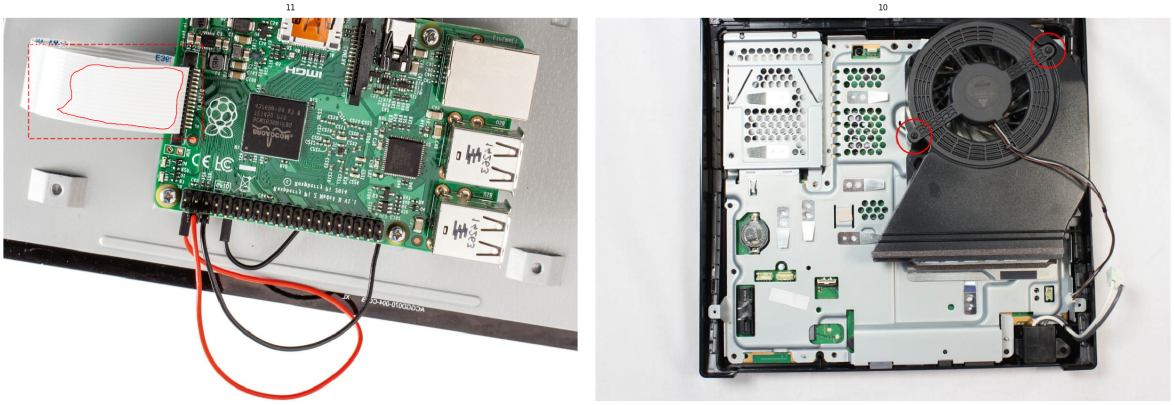

Figure A.20.: Cable detection in common setups.

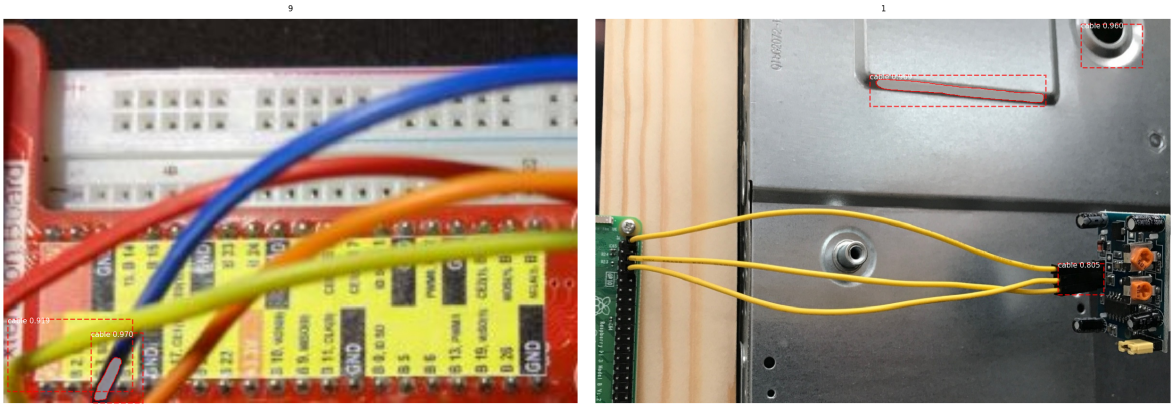

Figure A.21.: Cable detection in common setups.

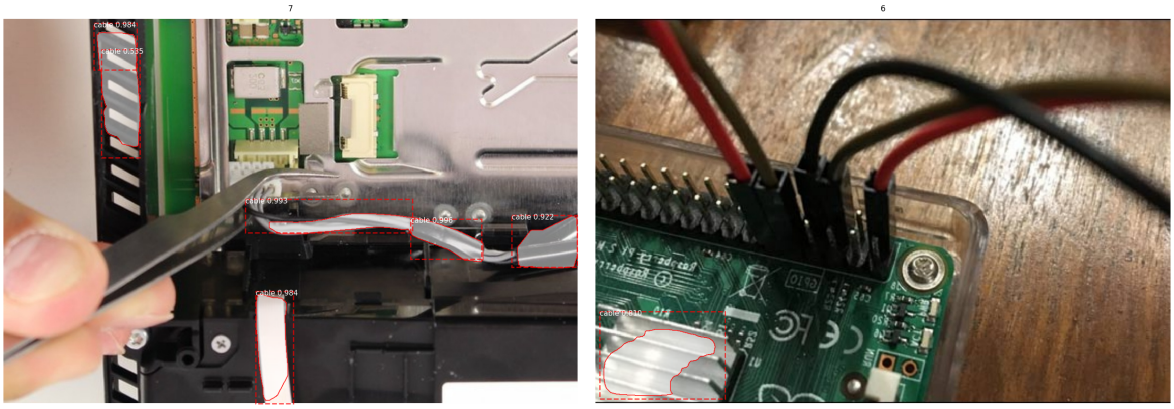

Figure A.22.: Cable detection in common setups.

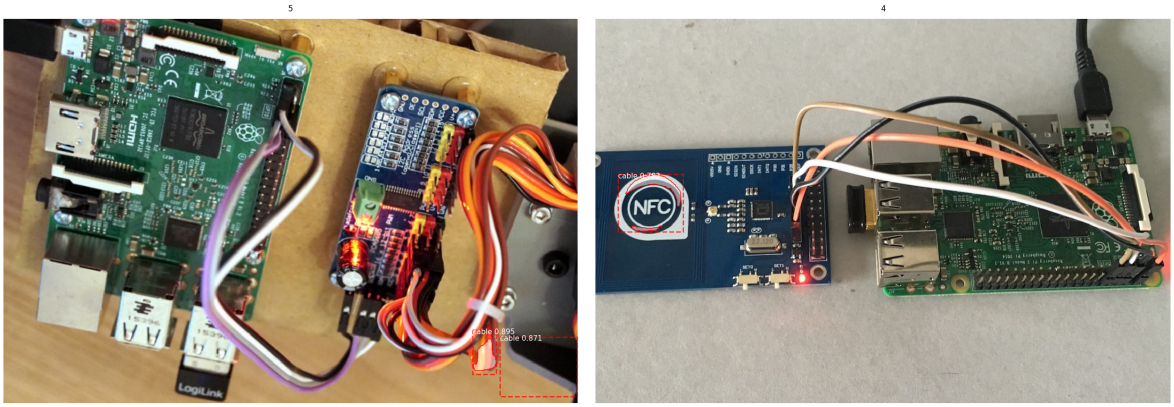

Figure A.23.: Cable detection in common setups.

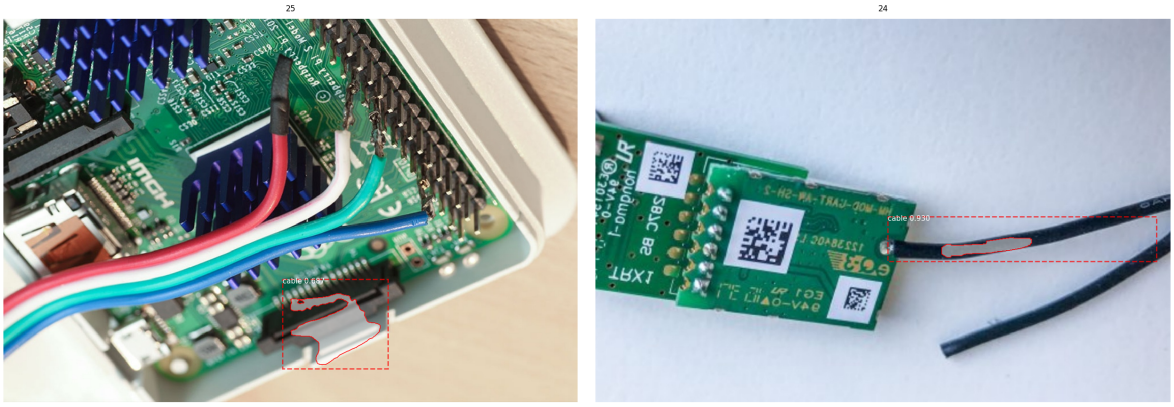

Figure A.24.: Cable detection in common setups.

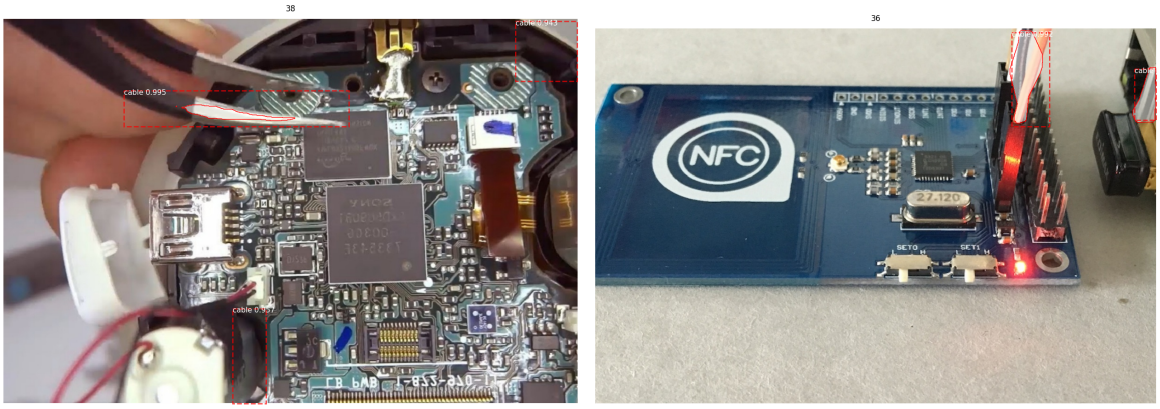

Figure A.25.: Cable detection in common setups.

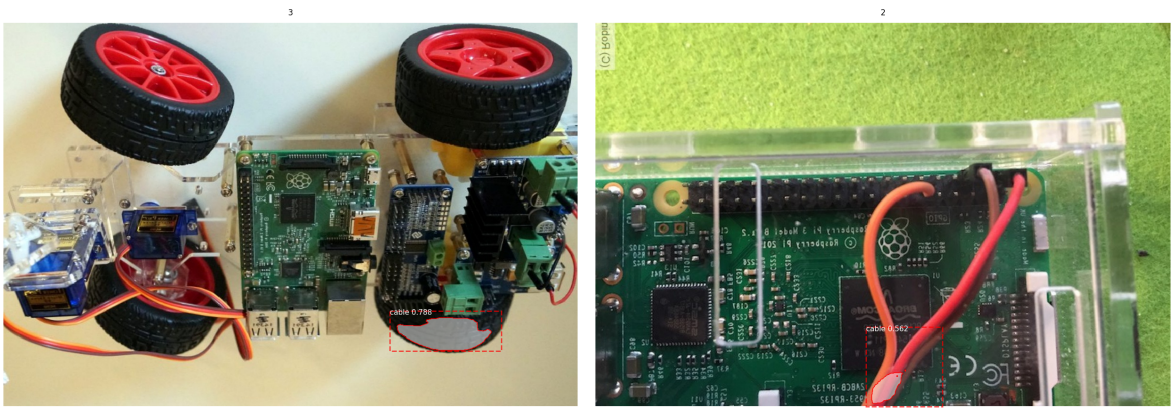

Figure A.26.: Cable detection in common setups.

Bibliography

[1] S. Ren, K. He, R. Girshick, J. Sun, *Faster R-CNN: Towards Real-Time Object Detectionwith Region Proposal Networks* (2012), URL `http://papers.nips.cc/paper/`

5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networ
pdf

[2] R. Girshick, *Fast R-CNN* (2015), URL http://openaccess.thecvf.com/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf

[3] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, S. Belongi, *Feature Pyramid Networks for Object Detection* (17 April 2017), URL http://202.119.32.195/cache/6/03/lamda.nju.edu.cn/9cb649a92e85a7ffb9624bd01aa35449/pr17.pdf

[4] K. HeGeorgia, G. Dollar, R. Girshick, *Mask R-CNN* (2018), URL https://arxiv.org/pdf/1703.06870.pdf

[5] P. Schumacher, M. Jouaneh, *A system for automated disassembly of snap-fit covers* (24 July 2013), URL https://link.springer.com/content/pdf/10.1007%2Fs00170-013-5174-8.pdf

[6] B. Kopacek, P. Kopace, *INTELLIGENT DISASSEMBLY OF ELECTRONIC EQUIPMENT* (1998), URL https://www.sciencedirect.com/science/article/pii/S147466701740262X

[7] R. Madaan, D. Maturana, S. Scherer, *Wire Detection using Synthetic Data and Dilated Convolutional Networks for Unmanned Aerial Vehicles* (2017), URL https://www.ri.cmu.edu/wp-content/uploads/2017/08/root.pdf

[8] R. Kasturi, O. I. Camps, *Wire Detection Algorithms for Navigation* (30 June 2002), URL https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20020060508.pdf

[9] W. Zhang, C. Witharana, A. K. Liljedahl, M. Kanevskiy, *Deep Convolutional Neural Networks for Automated Characterization of Arctic Ice-Wedge Polygons in Very High Spatial Resolution Aerial Imagery* (2018), URL https://www.mdpi.com/2072-4292/10/9/1487/htm

[10] X. Weia, C. Xiea, J. Wu, C. Shen, *Mask-CNN: Localizing parts and selecting descriptors for fine-grained bird-species categorization* (17 June 2017), URL http://202.119.32.195/cache/6/03/lamda.nju.edu.cn/9cb649a92e85a7ffb9624bd01aa35449/pr17.pdf

[11] L. Ding, A. Goshtasby, *On the Canny edge detector* (2005), URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.7086&rep=rep1&type=pdf

[12] R. Seising, M. E. Tabacchi, *A Very Brief History of Soft Computing: Fuzzy Sets, Artificial Neural Networks and Evolutionary Computation* (2013), URL https://www.researchgate.net/profile/Rudolf_Seising/publication/261300607_A_very_brief_history_of_soft_computing_Fuzzy_Sets_artificial_Neural_Networks_and_Evolutionary_Computation/links/5559f9d008aeaaff3bfab624.pdf

[13] R. Seising, M. E. Tabacchi, *Soft Computing* (2017), URL `https://www.sciencedirect.com/topics/computer-science/soft-computing`

[14] R. Rojas, *Neuronal Networks A Systematic Introduction* (1996), URL `https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf`

[15] H. Gao, L. Yixuan, G. Pleiss, *SNAPSHOTENSEMBLES: TRAIN 1, GET M FOR FREE* (2017), URL `https://arxiv.org/pdf/1704.00109.pdf`

[16] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht, *The Marginal Value of Adaptive Gradient Methodsin Machine Learning* (May 2018), URL `https://arxiv.org/pdf/1705.08292.pdf`

[17] S. Ruder, *An overview of gradient descent optimizationalgorithms* (June 15 2017), URL `https://arxiv.org/pdf/1609.04747.pdf`

[18] K. He, X. Zhang, S. Ren, J. Sun, *Deep Residual Learning for Image Recognition* (2015), URL `https://arxiv.org/abs/1512.03385`

[19] A. Jung, *Imgaug* (Jun 14, 2019), URL `https://imgaug.readthedocs.io/en/latest/#`

[20] S. Dodge, L. Karam, *Understanding How Image Quality Affects DeepNeural Networks* (21 Apr 2016), URL `https://arxiv.org/pdf/1604.04004.pdf`