Providing Automatic Instance Segmentation and Classification of Meal Images using Deep Neural Networks

Master Thesis

submitted by

Daniel Bogenrieder

(01/859162)

at the

Universität Konstanz

Faculty of Science

Department of Computer and Information Science

First Reviewer: Prof. Dr. Bastian Goldlücke
Second Reviewer: Prof. Dr. Harald Reiterer
Supervisor: Sebastian Hubenschmid

Konstanz, 2021

ABSTRACT

With obesity, diabetes, and in general, with the trend of having a healthier lifestyle, tracking food intake becomes more relevant. With the advance in machine learning and, especially, the improvements in deep learning, it is possible to analyze the food intake with a single image of the meal. This could help to automatically track different kinds of meals and also the amount of food. Even though this fully automatic detection might not be feasible yet, it is possible to provide a list of possible foods to the user and guess the amount of food. Various works have already tried to find a solution for this food detection, but food detection in images seems to be a particularly challenging problem. Pictures of the same food type can vary a lot depending on the used ingredients, the image quality, or the way of serving the food. Even more demanding is the problem that the different food items can occlude each other, making it hard to give a correct volume prediction.
Although there are many works that already cover similar tasks, this work focuses on getting the detection ready to be used with European dishes, provide food image segmentation in addition to classification and using the power of a state of the art Neural Network architecture. Therefore, this work introduces a pipeline, that fulfils the steps necessary to get from unlabeled food images to a trained neural network model, that can predict the food classes and the location of the food parts in the image. These steps consist of the creation of ground truth values, cleaning up and validating these values, training a neural network and deploying the result onto a server. Moreover, this automatic detection was integrated in an existing food tracking app. To evaluate the usefulness of this automation, a usability study with fifteen participants was conducted. The automatic detection speeds up the tracking time of the participants and has the advantage, that the food locations in the images are tracked as well. Based on the results of this study, further research directions and additional improvements of the system are presented.

PUBLICATIONS

Parts of this work are based on the documents:

> Daniel Bogenrieder. 'Segmentation and Classification of real-life food images'. Seminar to the Master Project. 2020.

and

> Daniel Bogenrieder. 'Creation of a Pipeline for Food Image Segmentation using Neural Networks'. Master Project Report. 2020.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

App	Application
WISeR	Wide-Slice Residual Networks for Food Recognition
SIFT	Scale Invariant Feature Transform
HoG	Histogram of Gradients
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
SVMs	Support Vector Machines
RoI	Regions of Interest
ADAS	Advanced Driver Assistance Systems
ReLUs	Rectified Linear Units
BP	Backpropagation
GANs	Generative Adversarial Networks
PFID	Pittsburgh Fast-Food Image Dataset
CNN's	Convolutional Neuronal Networks
COCO	Common Objects in Context
MTurk	Amazon's Mechanical Turk
CSV	Comma-Separated Values

1 INTRODUCTION

Eating is a crucial part of everyday life and plays an essential role in countering obesity and diabetes. Therefore, it is crucial to have a balanced and controlled food intake. To help people get a better overview of their intake, food tracking is becoming more relevant. Nevertheless, tracking food intake can be time-consuming. One needs to have a smartphone at hand and needs to input each of the individual parts of the meal. Additionally, if a calorie estimate is wanted, expert knowledge or special equipment is needed. That is why the United States Government has passed a law to force restaurants to label their meals with the number of calories[1]. This law gives the possibility to calculate the calorie intake when eating in a restaurant. However, it still does not help when cooking at home. To assist in tracking self-cooked food, applications have been created. BalancedEater is an example application developed as part of the SmartAct[2] project. With the help of this application, a picture of the dish is taken, and afterward, the different kinds of meals need to be selected out of a list (of meals). After that, an estimate of the meals' size needs to be provided. Figure 1.1 shows a storyboard of the described process.
This workflow is time-consuming and error-prone as users tend to underestimate their food intake [54]. To get a more objective reporting, automatic food detection methods have been researched. Kawano et al. proposed a system that uses image analysis methods and machine learning to train a classifier to detect food parts on which the classifier was trained on [37]. Bossard further improved this by making use of Neural Networks to get a more generic detection [5]. Both of these methods use Machine Learning to generate proposals of food types. Afterward, the user checks these proposals and, in the best case, accepts those. Although great success and accuracy could already be proven for general image classification [27, 42, 67], food image classification seems to be a particularly challenging task. This challenge results from the high intra-class variability of food classes. Intra-class variability means that the same class's food items can look different depending on the cook, lighting conditions, and camera quality. Furthermore, foods are typically deformable objects and therefore, it is hard to define a standard structure [51].

The first part of this work focuses on giving a more in-depth insight into the topic of food estimation. Furthermore, it aims to investigate the efficacy of ML algorithms to help users classify their food images more quickly. Therefore, in Chapter 2, the basics of Neural Networks and their most popular network architectures are presented. Afterwards, Chapter 3 presents the related work before in Chapter 4, the available datasets are shown.
The second part of this thesis deals with the implementation details of the process of food detection and classification. Therefore, Chapter 5 explains the creation of the

1 Menu Labeling Requirements, https://www.fda.gov/food/food-labeling-nutrition/menu-labeling-requirements, accessed: 19.07.2020
2 SmartAct website https://www.uni-konstanz.de/smartact/

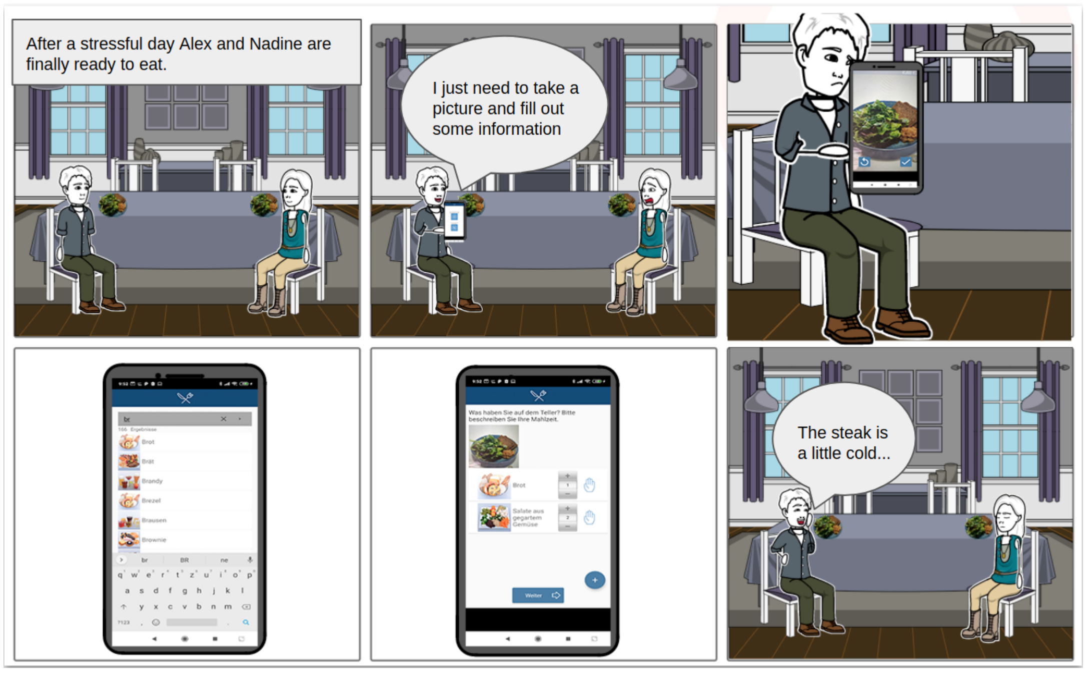

After a stressful day Alex and Nadine are finally ready to eat.
I just need to take a picture and fill out some information
The steak is a little cold...

Figure 1.1: This story board shows a common scenario of how the SMARTFOOD App is used. Alex and Nadine want to start having dinner, when Alex notices, that he has to take a picture of the meal. While the image is taken fast, he still has to search for the different food parts, which are in the image. This process takes some time and Nadine is not happy about it, because her food is getting cold.

basic pipeline used to generate a Neural Network model that can perform a food segmentation of images. Chapter 6 shows the needed enhancements that were necessary to support not only segmentation but also classification of the food images.
The third part of this work covers the details of the usability study. In Chapter 7 the study and the results of this study are described and discussed. Finally, in Chapter 8, the work is summarized and possible further steps are elaborated.
Although the focus of this work is to find all the food parts of a provided image, the created pipeline could be used for different tasks as well. These other tasks could be any task for which training images are given, and the wanted output is a segmentation of the input image. For example, this could be a brain-tumor detection on medical images or a car detection for autonomous driving.

2 BASICS

This chapter provides basic knowledge about the topics needed for the related work. Therefore, at first the different tasks are briefly explained. Afterwards, the concepts of Neural Networks and especially Convolutional Neural Networks are explained. After describing this basics, the most popular network architectures are presented in more depth.

2.1 SUITABLE TASKS FOR NEURAL NETWORKS

Neural Networks can be used for different tasks, such as image classification and image segmentation (see Figure 2.1). As each task has its own specific requirements and goals the relevant tasks for the further work are briefly explained in the following subsections:

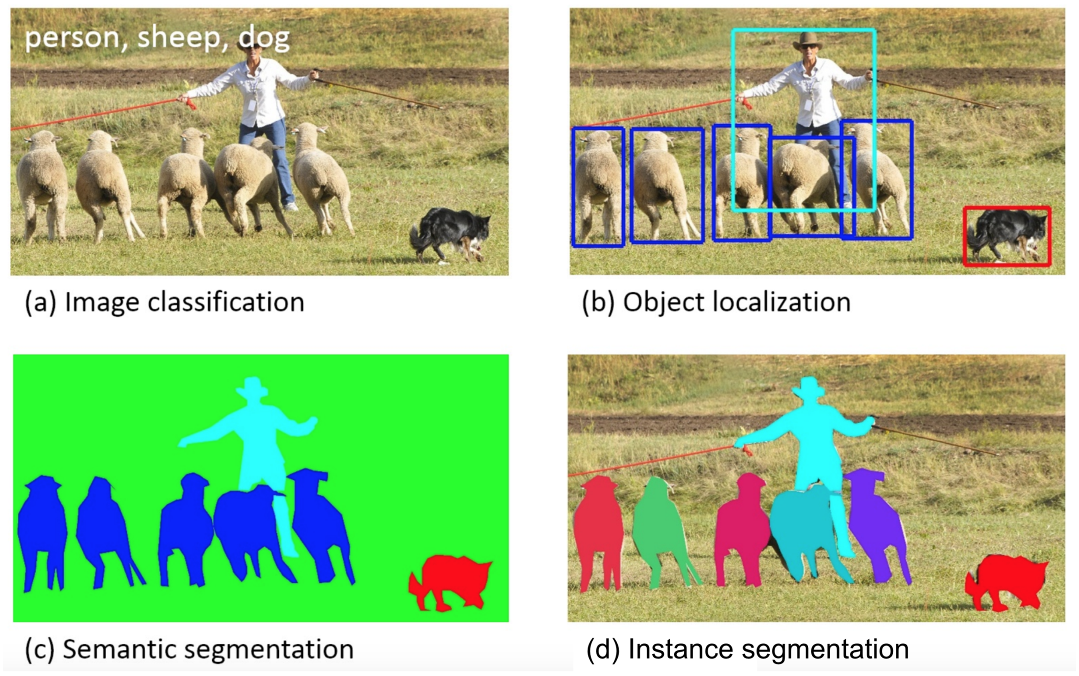


Figure 2.1: Sample images for the different tasks in computer vision. Image taken from [46]

2.1.1 *Classification*

Classification is the problem of finding the correct class from a predefined set of classes for a given input. This is shown in Figure 2.1(a) where the input image is given and the classes are given in the top left corner. Although in this example there are three classes provided, it is common that only one class is predicted. As Classification is an important topic in the field of Computer Vision, there are already various works that have focused on this problem and have proposed various ways to solve it [42, 65, 67].

2.1.2 *Object Localization*

Object detection refers to the task of detecting and also localizing an object. An example of this is shown in Figure 2.1(b). This task is a common problem in Computer Vision and has been the focus of research for a long time already. Analytical methods like Scale Invariant Feature Transform [47] or Histogram of Gradients [12] have been used in the early days to solve this problem. Since deep networks became popular, they have shown to be very effective in solving the problem and therefore a bunch of papers are already present and have shown the benefits of Neural Networks [20, 21, 52]. This includes the increase in prediction accuracy and also the removing the need for hand crafted features.

2.1.3 *Semantic Segmentation*

Semantic image segmentation, also called pixel-level classification, is the task of clustering parts of the image together, which belong to the same object class [69]. An example of this is shown in Figure 2.1(c). There are various types of applications of semantic segmentation. These include Advanced Driver Assistance Systems and self-driving cars [34], but it is also heavily used in the medical field. A common use case in the medical field is the detection of brain tumors [55] or detection and tracking of medical instruments during operations [19].

2.1.4 *Instance Segmentation*

Instance segmentation is closely related to semantic segmentation. It also works on a per-pixel basis. Both semantic and instance segmentation group together pixels that belong to the same class. Whereas semantic segmentation does not differentiate between individual objects, instance segmentation further differentiates between the single instances of the object. An example of this is shown in Figure 2.1(d). Instance segmentation is important for an automatic food intake tracking, as it is not only important to detect if and where bread for example is in the image, but it is also important to detect how many different slices of bread are present in the image.

2.1.5 *Volume Estimation*

In this work volume estimation is the problem of estimating the volume of an object based on a simple 2D image. This step is important towards calculating for example the calorie values of a food image. There are a lot of works tackling this problem for 3D reconstruction tasks [7, 17, 28] and also some for the specific task of food calorie estimation [18, 52].

2.2 NEURAL NETWORKS

Due to the proven effectiveness of Neural Networks, they have gained a lot of attention. In order to have a basic understanding on what Neural Networks are actual doing this section aims at giving a high level idea about Neural Networks. To start off, a short description of the history and general idea of Neural Networks is given. Afterwards, a high level introduction on how to train a Neural Network and also what kind of training methods exists is given. In the next section, an overview of typical applications for Neural Networks is provided. In the end, the benefits and drawbacks of using Neural Networks is discussed.

2.2.1 *The Perceptron*

The beginning of Neural Networks was heavily inspired by research trying to model the behavior of a human brain to achieve human learning capabilities [57]. These efforts resulted in the model, which can be seen in Figure 2.2. In this model, the dendrites (incoming branches of a neuron) are modeled as weights, which are multiplied with the incoming input values. An additional bias value is added to this value to mimic the activation potential of a neuron. Afterwards, all the incoming values from the dendrites are summed up and passed through an activation function. The output of the activation function can be interpreted as the neuron firing, thus contributing to the next neuron's input. The different activation functions exceed the scope of this work, therefore a step function will be used. Due to that activation function, binary a output is achieved, which makes the perceptron a linear classifier. The weights then span a hyperplane as a linear decision boundary between the different classes. This linearity limits the perceptron's power as, for example, an XOR function cannot be modeled by a single perceptron, as there can be no line to separate the two classes.

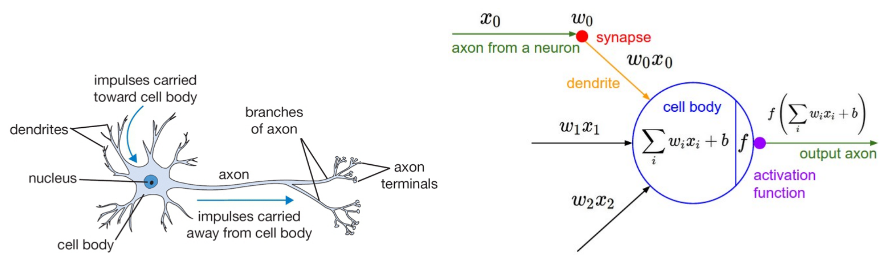


Figure 2.2: A simplified drawing of a biological neuron (left) and its mathematical model (right)[35].

2.2.2 *Neural Networks*

To be able to compute more complex functions like the XOR-function, Neural Networks were introduced. Neural Networks consist of multiple perceptrons that are connected in an acyclic graph. Most often, the architecture of the multi-layer perceptron is used, in which the neurons are grouped in layers. The first layer is then commonly called the input layer, whereas the last layer is called the output layer.

Every other layer in the middle of the input and output layer is called the hidden layer (see Figure 2.3). One tries to increase the complexity and capacity of the network with the use of these hidden layers. In general, the idea is that while training, each perceptron adapts to another specific characteristics of the input data (For example the presence or absence of an edge in an image). The combination of these different characteristics then result in different classes. It can be shown that a Neural Network with at least one hidden layer is already enough to function as a universal approximator – that is that any arbitrary function can be approximated to an ordinary precision by this network if there are sufficient neurons in that hidden layer [32]. This fact might sound like one hidden layer is enough, but the problem with this theorem is that the word "sufficient" means an exponential number that is unfeasible to have. Therefore with an increase in depth (amount of hidden layers), fewer neurons are needed. Thus far, there is no generic formula which states the optimal depth of a network, but research has shown, that deeper networks perform better [3, 23, 67].
There are two main problems when it comes to choosing the right amount of neurons. If too many neurons or layers are chosen, then overfitting might happen. Overfitting is the problem that the network learns the exact structure of the training data and thus loses its ability to generalize to new data [42]. If the network performs well on the training dataset, but not good on the validation dataset, the reason is probably overfitting. The other problem is the problem of underfitting, which is that the complexity of the network is too low to map the underlying function. The method to encounter underfitting is obviously to increase the amount of training data provided. Countering overfitting is harder, as it cannot be seen directly from the performance of the network. There are multiple approaches to counter it. One very important approach is called dropout and was introduced in the AlexNet network [42]. Each neuron of the network has a predefined probability to be disabled for a specific training sample while training the network. Thus the network cannot rely on specific features to be present in the network and has to build a more robust method of finding the correct class.

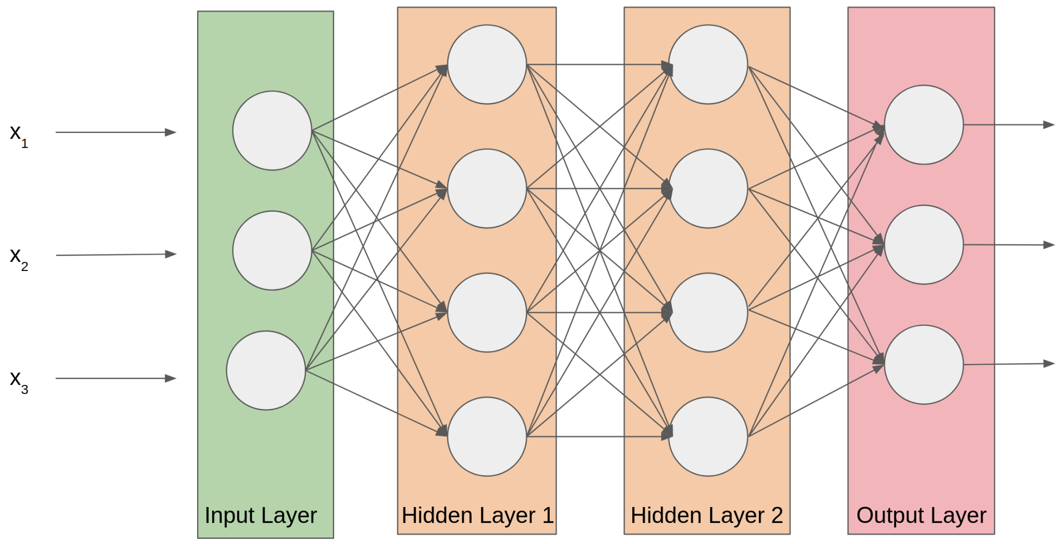


Figure 2.3: A 3-layer Neural Network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that there are connections (synapses) between neurons across layers, but not within a layer.

2.2.3 *Training a Neural Network*

The good thing about Neural Networks is, that the network tries to learn the parameters itself. In order to achieve this, a lot of data needs to be provided to the network.

To train a Neural Network, all the weights and biases of the hidden layers are iteratively adjusted, so that the network can approximate target outputs from known inputs. Due to the large number of parameters and the high complexity of the network, it is difficult to solve these parameters analytically. Therefore, the back-propagation algorithm [59] was proposed. This algorithm consists of multiple phases. In the first phase of the algorithm, a vector of the inputs is propagated forward through the network. The received output is then compared to the known ground truth data, and with the help of a loss function, the difference of the calculated output and the ground truth data is calculated. There are various loss functions out there, which will not further be discussed here. For simplicity, it is assumed the loss function is the mean square error function, which is shown in Equation 2.1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

The gradient, a mathematical construct in order to find a minimum, of this loss function is then computed. In the case of using mean squared error as the loss function, the output of the output layer is then simply the difference of the calculated and the ground truth data. The second phase starts, in which the gradient is propagated back through the network to calculate the gradient values of the hidden layer neurons. This calculation is done by using the chain rule of derivatives. Afterward, the weights are updated by calculating the gradients of the weights and subtracting a proportion of the gradient from the weights. This proportion is called the learning rate, which is a predefined parameter. It is common practice to adjust the learning rate every few iterations dynamically. When the last layer has been updated, the next input is chosen, and the process is repeated until the weights converge.

2.2.3.1 *Supervised Learning*

It is called supervised learning, when a trainig dataset is present and for this dataset the solution (ground truth data) is also known. The algorithm then iteratively proposes a solution which is then checked against the ground truth data and refines its decision making if the output was not correct. This type of learning is the most common learning type in classification problems.

2.2.3.2 *Unsupervised Learning*

Unsupervised learning is when only unlabeled data is provided, and the algorithm is left to itself when it comes to finding the underlying structure of the data. This type of learning is mostly common in clustering tasks. Autoencoders are a popular kind of Neural Networks that use unsupervised learning. The idea behind Autoencoders is that they learn to map the input to the output. The network, therefore, consist of two parts. An encoder reduces the dimensionality of the input data and a decoder, which decodes this feature representation of the encoder and tries to recover the original input. A simple example would be to have a cat image as input data. The encoder would then encode this image representation into a feature representation, while the decode tries to recover the cat image with the help of this feature representation. Once the training is completed, one could remove the encoder part and generate new cat images from using randomly generated feature representations. This method is nowadays prevalent and specially used in generative models when it comes to generating new content.

2.2.3.3 *Online Training*

The method, in which the weights are directly adjusted after every training sample, is called online training. As every sample can be arbitrary different, this can lead to a jumping behavior. As the optimal result of the network is the minimum of the loss function, this jumping behavior makes it hard to find that minimum.

2.2.3.4 *Batch Training*

In contrast to the online training, there is the full batch method, which uses the complete training set before adjusting the weights. This approach is computationally heavy because, for every adjustment, the algorithm needs to look at the complete training set. Hence this approach results in long training times. Therefore a variant of the batch training is proposed, which is called mini-batch training. As the name suggests, it groups the training data into small subsets that are processed before adjusting the weights. These groups are redefined on every iteration of the training in order to remove side effects.

2.2.3.5 *Transfer Learning*

Training a Neural Network can be very time consuming and requires lots of data. However, in general, the hope of the models of Neural Networks is that the lower layers learn low-level details of the input and the higher level layers lear more abstract features. In images, for example, these low level features could be edges or lines. As these features should generalize well to other tasks, a common approach is to fix some parts of the network and only train some higher layers of the network. Of course, this means that the more similar the primary task was to the new task, the better this works. It is common practice to use transfer learning on models that have been pre-trained on large datasets like the Microsoft COCO or ImageNet dataset [29].

2.2.4 *Applications of Neural Networks*

One of the early applications of Neural Networks was to identify handwritten digits and letters. In the early 1980s, good results could be achieved by using Neural Networks for this task [11]. Although Neural Networks experienced a lot of attention in the 80s, this exitement calmed in the 90s and the focus was shifted towords Support Vector Machines (SVMs). With more powerful computers and the use of dedicated graphic units, Neural Networks became popular again when the AlexNet won first place on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014[1]. The ILSVRC was one of the most important challenges between 2010 - 2017. ImageNet provided a huge dataset of images and two tasks. There was the task of object detection and the task of object classification. Due to the massive investments in this field after the success of AlexNet, Neural Networks were adopted in many fields like image classification, image segmentation, and prediction tasks. Figure 2.4 shows the winners of the ILSVRC from 2010 to 2019. The two winners from 2010 and 2011 used analytical methods, whereas all the winners, after that used Neural Networks in order to achieve the shown results. As can be seen, this led to a large reduction of the error rate. The details of the different architectures are discussed in section 2.3. As a side result of their paper Russakovsky et al. [60] tried to measure the error rate of a human on such a classification task. The result was approximately 5%. As ResNet achieved an error rate of only 3.57%, a network achieved better an accuracy than a human, although this value needs to be handled with caution. There were multiple problems when this value was measured. Only a few people were tested to measure the performance, and another problem was that the task is monotonously and therefore boring for a normal person. This resulted in a bad performance when the task has many samples, but the human could still be better than the network if the task has only a small amount of images. Nevertheless, this shows that Neural Networks can beat humans in some specific tasks. That humans are not unbeatable was shown even more popular with the more recent winnings of the Neural Network Alpha Go against a grandmaster GO-player [2].

1 http://www.image-net.org/challenges/ILSVRC/
2 https://www.theverge.com/2019/11/27/20985260/ai-go-alphago-lee-se-dol-retired-deepmind-defeat

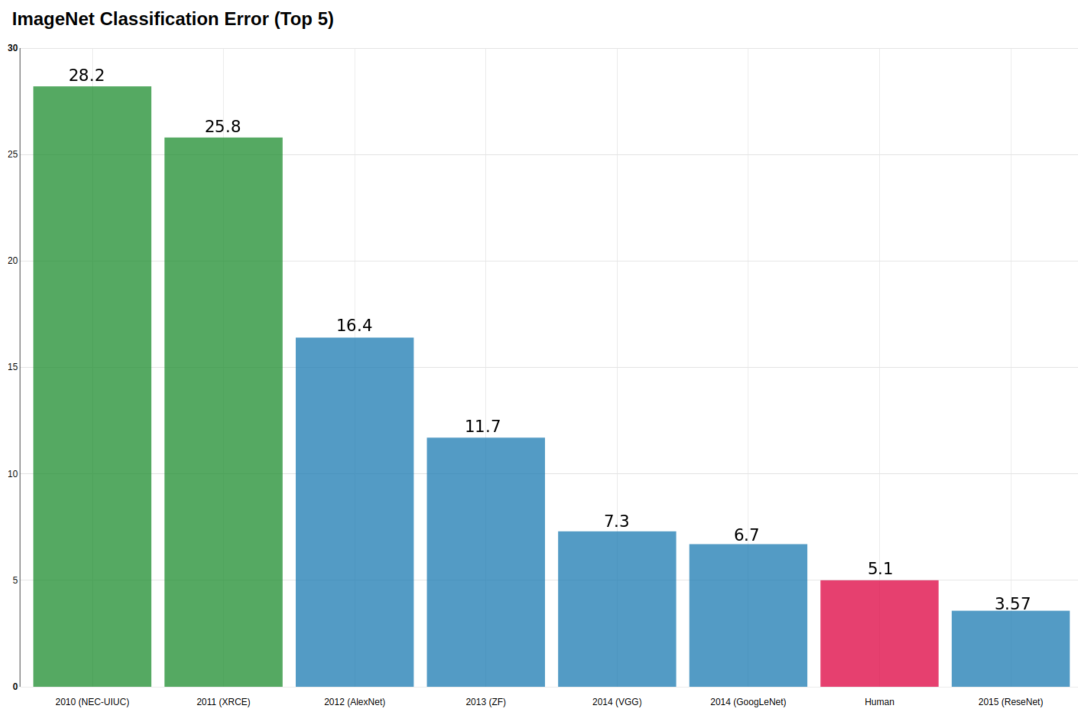


Figure 2.4: Graphical representation of the winners of the ILSVRC- Challenge.

2.2.5 *Benefits and Drawbacks of Neural Networks*

Although Neural Networks have shown great success in recent years, it is essential to also keep the drawbacks in mind. Therefore this subsection focuses on the benefits and drawbacks of Neural Networks. A summary of this analysis is shown in Table 2.1.

Neural Networks can be used to model complex functions. To be able to approximate such a function, the model needs to be trained. Although there is no formula known on how much data is needed, it is known that the more complex a function is, the more training data is needed. In the ILSVRC, for example, 1.2 Million images were provided in order to classify 1000 categories. The problem is not only that there is a need for a massive amount of images, but that these images need to be labeled in order to train the network. As it is inhumane to label 1.2 million images manually, several methods, such as crowd sourcing services, have arisen in order to collect this data. Another problem is, that much time is needed to train a Neural Network with such a massive amount of data. Special frameworks like NVIDIA's CUDA[3] and modern GPU's led to a considerable speed-up of the needed training time. However, even with the use of multiple modern GPU's, training can take a long time. VGG, an important network architecture, took on 4 NVIDIA Titan GPUs from 2 to 3 weeks (depending on the network configuration) [67]. This means that even with an expensive hardware training time of a Neural Network is still long. One of the goals of Neural Networks is that even if the training time is long, when the model is deployed, the predictions should be rather fast. If trained correctly, the

3 https://developer.nvidia.com/cuda-zone

models can be very accurate and even surpass a human [60]. Another advantage is that there is no need for handcrafted features. Once the architecture is set up, the network is supposed to learn the weights by the training images and, therefore, also learn the features itself. Although this might be an advantage on the one hand, because it can get complex to find suitable features, it also makes it hard to see why the network has chosen a specific output. This can be very important when it comes to e.g. medical use cases.

Benefits	**Drawbacks**
+ Neural Networks can map very complex structures	- Need a massive amount of labeled data
+ Fast prediction	- Need costly hardware
+ High accuracy	- Need much training time
+ Features do not need to be handcrafted	- Hard to see what exactly is going on

Table 2.1: Benefits and drawbacks of using Neural Networks

2.3 MOST POPULAR NETWORK ARCHITECTURES

In this section, an overview of the most popular network architectures is given.

2.3.1 *Layers*

To understand the network architectures at first an introduction to the different layers of a Neural Network is given.

2.3.1.1 *Convolution Layer*

Convolutional Neural Networks are a special group of Neural Networks. The idea is that not every neuron needs to get the input of all previous neurons. Therefore so called convolutional layers are introduced. The convolutional layer is the core block of a Convolutional Neural Networks that does most of the computational heavy lifting. It is used in order to reduce the number of needed parameters of each layer drastically. This reduction is accomplished by removing the need to connect every neuron of a layer to every neuron of the next layer. Without this reduction, the parameters can easily explode when considering high dimensional images. If, for example, images with a size of 800 x 600 pixels are considered, this would already result in 800 * 600 * 3 = 1.440.000 Weights for each neuron. Furthermore, as more than one neuron is probably wanted, this number grows very quickly and would require an unfeasible amount of computational power. Besides, if the input consists of images, it probably makes sense to look at pixels and their neighborhood rather then the complete image to find interesting insights. For that reason, research discarded the brain analogy, and looked more into how the visual coretex of humans and animals is working. Especially in images, it makes much sense not to flatten the input image and look at single pixel values but instead look at the neighbourhood of that pixel. This neighbourhood of a pixel builds an image region and is called the receptive field. It consists of the pixel value in the center, and a predefined neighborhood of the pixel. This idea is realized by using a convolution operation on the receptive field and a specified kernel. This kernel consists of a predefined height and width and the same depth as the input image. In a normal image, the depth than would be three, as there are the three color channels (red, green, blue) of an image. To calculate the activation of a particular neuron, a pointwise multiplication of the kernel weights and the image region is calculated. This operation is then done in a sliding window approach to every pixel of the image. An example of how this is done can be seen in Figure 2.5. The output of this operation is the so-called feature map. The dimension of this feature map depends mainly on three factors which are discussed in the following:

1. The depth of the output volume is a hyperparameter and corresponds to the number of different filters one would like to use. Hyperparameters are parameters that are not tuned while training the network and need to be defined beforehand. The idea is that each of these filters learns to look at something different. For example, in a lower layer, one neuron could activate depending on if there is a horizontal edge present in the image.

2. It needs to be defined how the filter is slid over the image. This hyperparameter is called stride and defines how many pixels the filter is moved to each step. In the example in Figure 2.5 the stride is chosen to be one, and therefore each step the filter is moved by one pixel. When the stride is chosen larger than one, then the filter jumps more then one pixels at a time. Doing that results in a reduction of the height and width of the output.

3. It depends on the method of how boundary pixels are treated. There are various methods to deal with this case. One could do a zero-padding in which the boundary is extended with zeroes to calculate the convolution at border pixels. Another approach would be to extend the image with the values of the border pixels. Otherwise, if the image is not padded, the output is reduced in size.

To further reduce the needed amount of parameters, a convolutional layer on images uses parameter sharing. "Parameter sharing works on the assumption that if one region feature is useful to compute at a set spatial region, then it is likely to be useful in another region. If we constrain each individual activation map within the output volume to the same weights and bias, then we will see a massive reduction in the number of parameters being produced by the convolutional layer." [53] (see Figure 2.6)

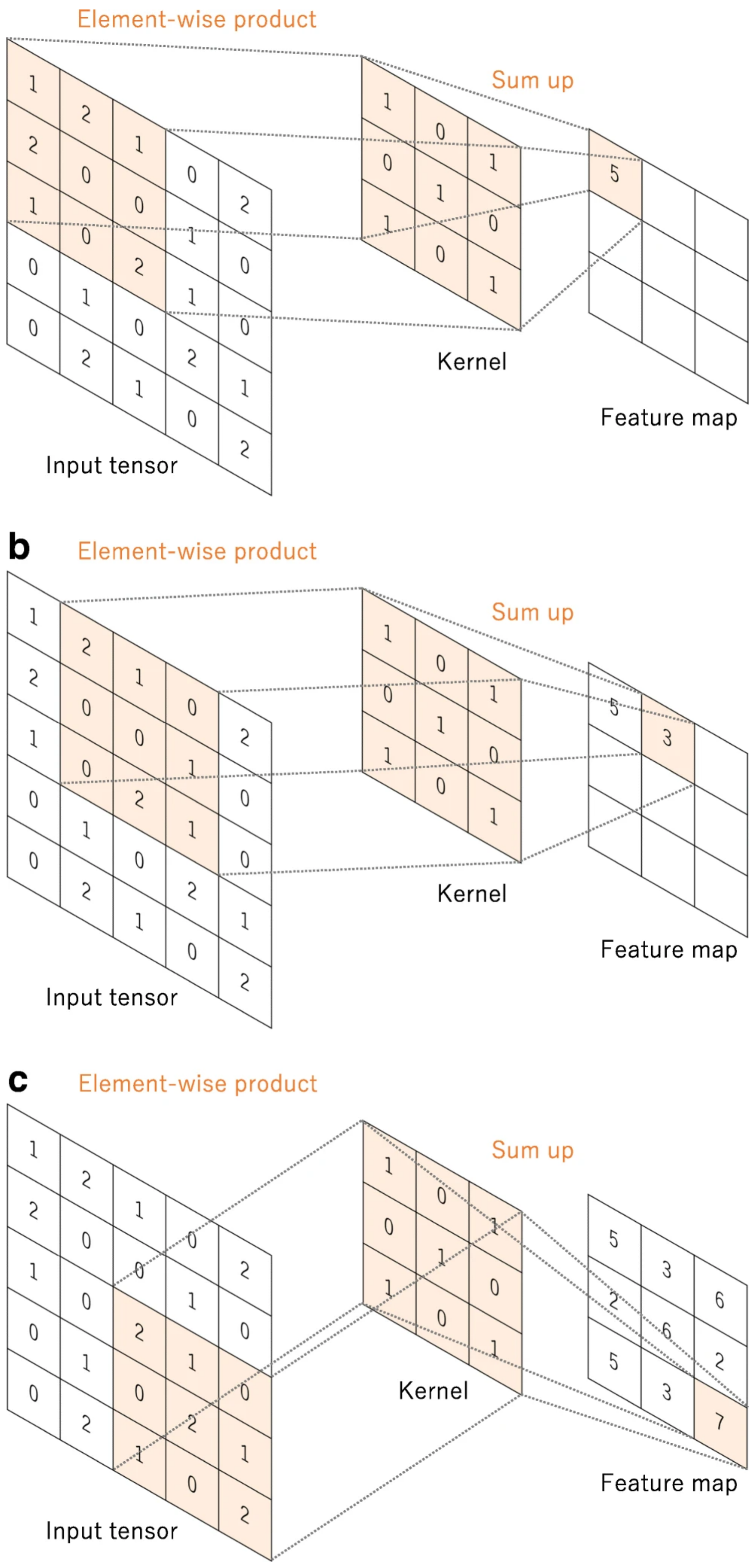


Figure 2.5: An example of the convolution operation with a kernel size of 3 x 3 with no padding and a stride of one. A kernel is applied across the input tensor, and an element-wise product between each element of the kernel and the input tensor is calculated at each location and summed to obtain the output value at the corresponding position of the output tensor, called feature map. [73]

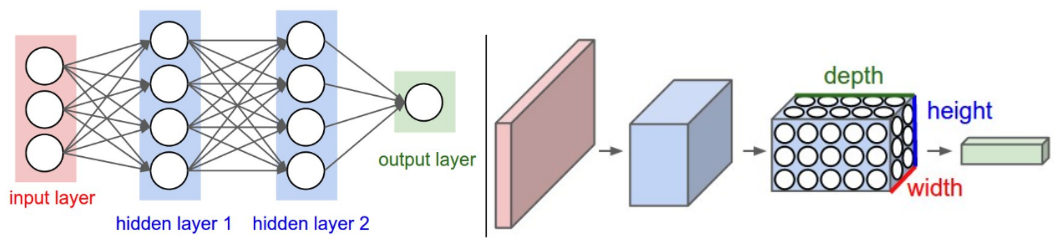


Figure 2.6: Left: Layout of a 3-layer Neural Network. Right: Structure of a Convolutional Neural Networks. The network arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Each layer of a Convolutional Neural Networks (CNN's) transforms the 3D input volume to a 3D output volume of neuron activation's. Image taken from [10]

2.3.1.2 *Pooling Layer*

The pooling layer aims to reduce the dimensionality of the input and also the number of parameters and therefore, the computational complexity of the model. The Pooling Layer operates on every single activation map and often uses the "MAX-function to achieve the reduction (see Figure 2.7). Many CNN's, therefore, use a max-pooling layer with a kernel size of 2x2 and a stride of two. This means that the kernel is looking at a 2x2 field in the image and takes only the highest number of this field. This pooling results in a reduction of parameters of 75%. Due to the destructive nature of the pooling layer, an increase of the size of the kernel or the stride is uncommon and most often results in a worse performance. Instead of the max function, other functions could also be used, such as an average function. Although the average function was often used historically, it has recently shown that the max function works better in practice [73].

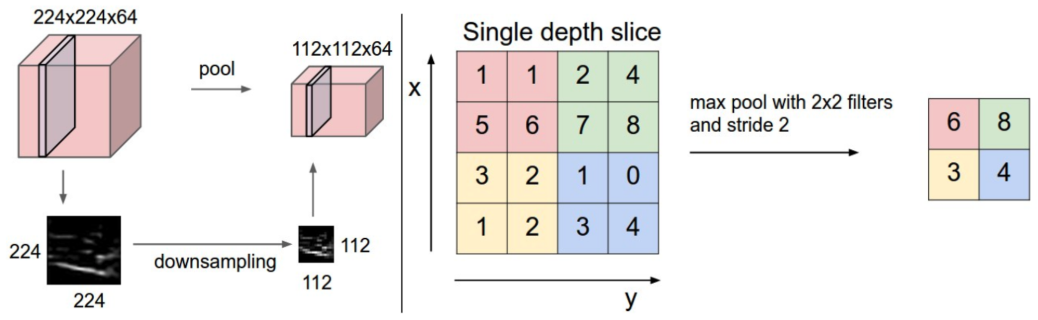


Figure 2.7: Left: Example of a pooling layer which acts on each depth slice independently and is used to reduce the size of the input. Right: A pooling operation is shown for a single depth slice with a 2x2 filter and a stride of 2 with the help of max pooling. [10]

2.3.1.3 *Fully connected layer*

All of the neurons of one layer are connected to all of the neurons of the next layer. Fully connected Layers are often used as the last layers of a Convolutional Neural Networks. The previous convolution- and pooling layers can be interpreted as feature extracting layers. The function of the fully connected layers is to find a possibly non-linear combination of these features to separate the different classes. Most of the parameters of a convolution Neural Network are based in those fully connected layers, in the end, to be able to learn these complex functions. For example, AlexNet has around 60 million parameters, out of which 58 correspond to the fully connected layers [62].

2.3.2 *Architectures*

After understandig the basic blocks of a Neural Network, in the following the most popular architectures will now be presented.

2.3.2.1 *AlexNet*

AlexNet, the first deep Convolutional Neural Networks named after the author Alex Krizhevsky was also the Neural Network that restarted the hype around Neural Networks. This started in 2012 when AlexNet achieved an improvement of around 12% in the ILSVRC. This improvement was achieved by various factors, which are explained in the following.

The architecture of AlexNet is depicted in Figure 2.8. It consists of eight learned layers. Five of these layers are convolutional- and three are fully connected- layers. Most of these layers are split into two branches. These two branches are needed to distribute the network's work onto two graphic cards, as one graphic only had 3 GB of memory and, therefore, not enough memory to fit the network.
Another significant improvement that was made in AlexNet was to use ReLUs as an activation function instead of a tanh or sigmoid functions. ReLUs have an essential advantage which is, that they are not limited to some number, and therefore do not saturate. This results in an increased training speed. Although this property of ReLUs reduces the need for input normalization, they still propose a local response normalization in the paper. Furthermore, they claim that this way, they improved the top-1 and top-5 error rates by 1.4% and 1.2%, respectively. In newer architectures, this kind of normalization is not used anymore. It has been shown that the influence of these normalizations tends to be minimal but increases memory consumption and computation time [65].
Besides, they found that with such a huge network (60 million parameters and 1000 classes), overfitting became a big problem and therefore proposed two important methods in order to counter overfitting. The first idea is to use data augmentation to increase the available dataset. Data Augmentation is a technique, where various image transformations are transformed to get additional training images and also train for skewed images. The other important method is called dropout and is still used in modern networks. Dropout was first introduced by Hinton et al. [30]. They showed in their paper that if neurons are deactivated with a probability of 0.5, this reduces overfitting. The idea behind this is that neurons can not rely on the presence of a particular previous neuron. The neuron is forced to learn more robust features that are useful in combination with many different random subsets of other neurons. In AlexNet dropout is only used in the last two fully connected layers and

only while training. When the network is at test time, all the neurons are used, but the output is multiplied with 0.5, which should compensate for the dropout.

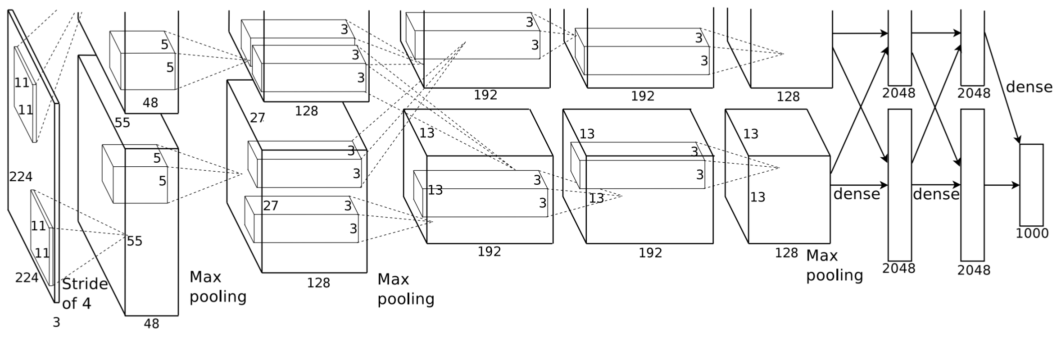


Figure 2.8: Architecture of the famous AlexNet [42]

2.3.2.2 *VGG*

VGG was published two years later then AlexNet in the ILSVRC-2014. While the winners of the ILSVRC-2013 won the challenge mainly due to hyperparameter tuning of AlexNet, Simonyan and Zisserman proposed multiple networks with an increase of depth of the networks. If simply the depth of the network is increased, this results in a huge increase in parameters. To counter this problem, Simonyan and Zisserman came up with an idea of using only small convolution filters of size 3x3 but stacking these. In contrast, AlexNet used a size of 11 x 11 in the first layer and a size of 5 x 5 in the second layer and 3 x 3 in the following layers. Stacking two 3 x 3 layers results in an effective receptive field of 5 x 5. Furthermore, when then stacking three 3 x 3 convolutions, it has an effective receptive field of 7 x 7. The benefit of doing these stacks of convolutions instead of a single bigger convolution is that instead of a single rectification layer, three rectification layers are used, making the function more discriminative [65]. Besides, the number of parameters needed is drastically reduced. In their paper, Simonyan and Zisserman show, that doing the bigger convolution results in 81% more parameters. Due to that trick, they were able to increase the depth of the network. It still needs to be kept in mind that VGG-16 has 138 Million parameters compared to 60 Million from AlexNet. This increases in the depth, and therefore the increase in parameters results in larger training time. While the authors of AlexNet claimed to have trained the network in 6 days on two GTX 580 3GB GPUs, Simonyan and Zisserman claim that the training time is 2-3 weeks on four NVIDIA Titan Black GPUs which should have 6GB of memory. A comparison of the architecture of the AlexNet and the VGG-16 architecture can be seen in Figure 2.9. Despite the big improvement in terms of accuracy they showed in their paper that the network generalizes well to a wide range of tasks and datasets.

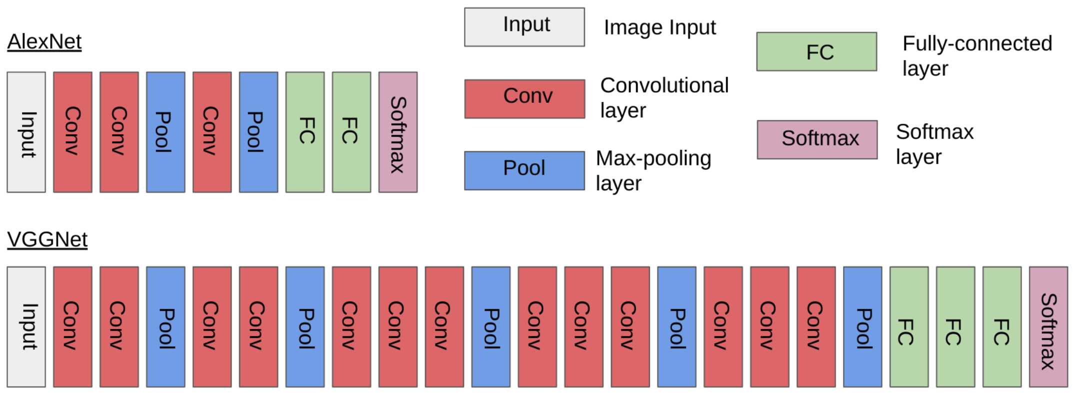


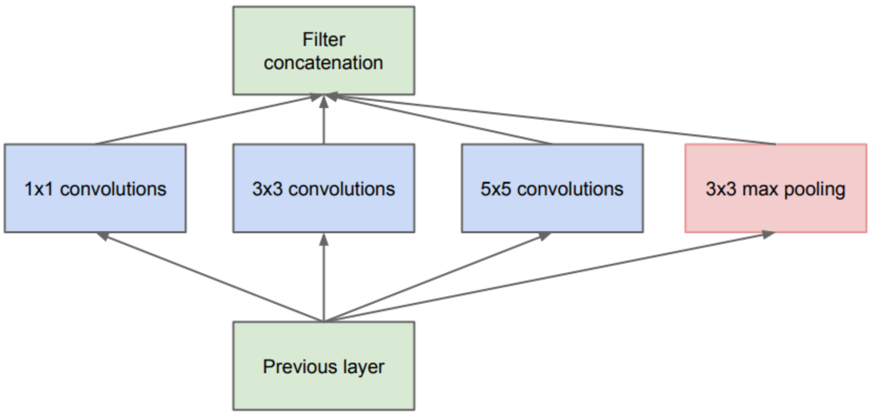
Figure 2.9: Comparison of the VGG-16- [65] and the AlexNet- architecture. Top: The structure of AlexNet, with its seven hidden layers, is displayed. Of those seven layers, five contain trainable parameters. Bottom: Structure of the VGG-16 architecture. It consists of 21 hidden layers, of which 16 are trainable. Taken and adapted from [36]

2.3.2.3 *GoogleNet*

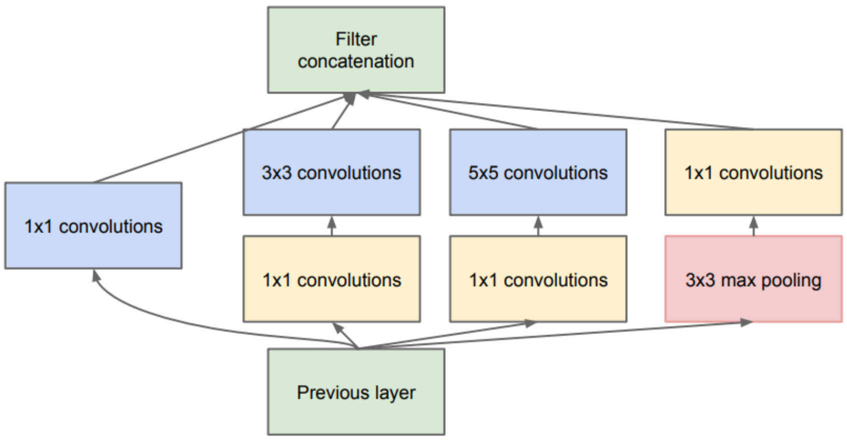
Although published in the same year as VGG, GoogleNet is quiet different to VGG. At the beginning of their paper, they state that the main purpose of the network is to make it efficient in order to make it possible to run on a mobile device. Of course, the goal was still to have high accuracy. In order to achieve those two requirements, they had to come up with a few tricks, as better accuracy was mostly achieved by deeper networks, but that also results in more parameters and, therefore, an increase in needed computational power, which is exactly what was not wanted. Inspiration was found in the paper of Lin et al. [45]. The idea of networks in networks was adopted further optimized. This resulted in the idea of inception modules. Instead of doing a single convolution on the image, they stacked multiple parallel convolution layers with different filter sizes and concatenated the result. With this, the network should be allowed to extract important features that might only be visible in a bigger or smaller receptive field. A model of the naive and improved version of this inception module is provided in Figure 2.10. The problem with this naive approach, especially when it is stacked, is, that it would completely blow up the necessary parameter amount. Therefore they introduced 1x1 convolutions before doing the computational heavy 3x3 and 5x5 convolutions. With the help of this 1x1 convolution, the dimension of the input can be reduced and thus be kept to a bearable size. Due to several techniques (e.g., 1x1 convolutions) [67], they were able to reduce the required parameters by a factor of 12 compared to AlexNet. While AlexNet needed around 60 Million parameters and VGG-16 even needed 138 Million parameters, GoogleNet only needs about 5 Million parameters and achieved better results in the ILSVRC-2014 then VGG.

2.3.2.4 *ResNet*

The previous two networks have already shown that it is common to increase the depth of the networks in order to increase the network's accuracy. In the first part



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2.10: Illustration of (a) a naive inception module and (b) an inception module with dimensionalty reduction [67]

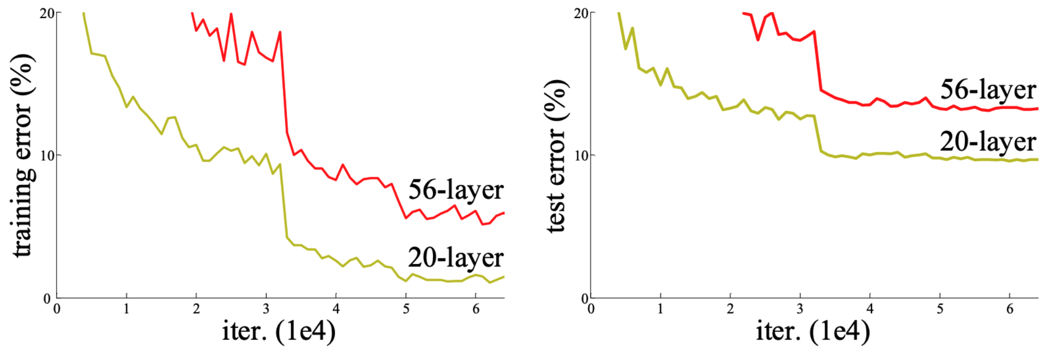


Figure 2.11: Training and test error of Neural Network with 20- and 56-layers. One would expect, that the 56-layered network performs better, or at least that there is some sort of evidence for overfitting [27].

of their paper, He et al. show that this is only true to some extend. In Figure 2.11, it is shown that a 56-layer network performs worse than its 20-layer counterpart. The reason for this is not overfitting. If overfitting was the problem, then the training error should not increase when increasing the depth of the network. When theoretically thinking about this problem, one can easily construct a network that should at least perform as good as the shallower network. If the added layers would do an identity mapping, then the network should at least perform as good as its shallower counterpart. However, as shown in Figure 2.11, current solvers are not able to find this solution (at least not in a reasonable time). With this in mind, He et al. came up with the idea to introduce residual blocks. These blocks introduce skip-connections, which bypass some convolutional layers and add the original input onto the output of the convolutional layers. An illustration of such a network architecture can be seen in Figure 2.12. The dashed skip connections indicate an increase in dimensions. In the paper, they tried three different methods in order to deal with that.

1. Zero padding shortcuts are used for increasing dimensions
2. Projection shortcuts are used for increasing dimensions, and other shortcuts are identity
3. All shortcuts are projections

They found, that while (1) was the worst and (3) was the best they still recommend to use (2) as it improves (1) without additional parameters or computation and (2) is only marginally worse than (3) and (3) would need additional parameters.
With the use of these residual blocks, they were able to increase the depth of the network by far. On ImageNet, they tried networks of up to 152 layers, which is 8x deeper than VGG nets. Additionally, they tried a network with over 1000 layers. This network had a very low training error, but on the test set, it performed worse than the 152-layered network. They argue that this is due to overfitting.

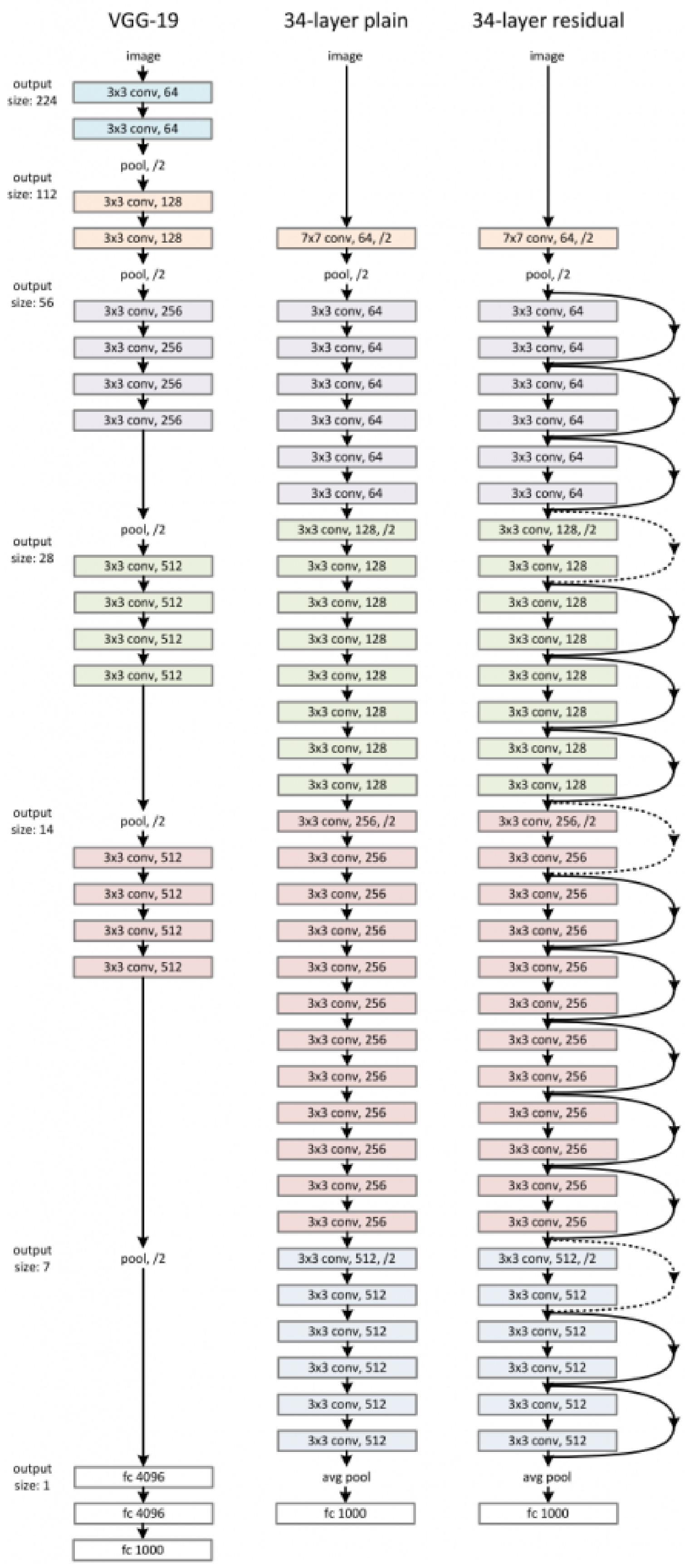


Figure 2.12: Comparison of VGG-19 and Resnet with 34-layers and with and without residual connections. [27]

2.3.2.5 *R-CNN*

For now, all the architectures presented are used for single instances, but there could also be images that contain the same object multiple times. The simplest approach might be to use a sliding window approach to get multiple image slices that might contain different objects and can then be used by one of the already presented architectures. Obviously, this would depend on the chosen window size and results in a large number of possibilities. A better approach is presented by Girshick et al. in 2014 [21] with the R-CNN algorithm. Regional-Convolutional Neural Networks comes from the fact that they used CNN features on different regions of the image. The main idea of the algorithm consists of three phases, which are also depicted in Figure 2.13:

1. Extract region proposals
2. Compute CNN features
3. Classify these CNN features

In the first phase, the algorithm extracts some "interesting" regions in the image. The meaning of interesting in this context is that it contains an object. This could be a simple sliding window approach, as described at the beginning of the section, but they used Selective Search to achieve this that. Selective Search is an algorithm that has a bunch of interesting regions as result. With the help of Selective Search, 2000 region proposals are produced. Each of these proposed image regions is then warped into a format that the CNN expects (in the paper 227 x 227 pixels). The second phase is then done using a CNN in order to do the feature extraction. In the paper, they used AlexNet (without the final softmax layer) as a backbone, but any of the previously described architectures could be used. In the third phase for each class, they score each extracted feature vector using the SVM[4] trained for that class. To improve the bounding boxed, a regression model is trained to correct the predicted detection.
Although this algorithm works quite well, it has three main problems, which makes it slow. The first is that running Selective Search in order to get around 2000 proposals is slow. The second is that the algorithm computes a CNN feature vector for each of the proposed regions separately, although they might overlap. The third is that the algorithm consists of three more or less separate models that do not share computation (the CNN for feature extraction, the SVM classifier for identifying objects, and the regression model for bounding box optimization).
With the two papers "Fast R-CNN" [20] and "Faster R-CNN" [56], these problems got tackled. Fast R-CNN mainly focused on solving the second and third problem, by performing the CNN part, not for each region individual, but performing it once for the whole image and then using the Region of Interest (RoI), which are still provided by Selective Search, from these feature maps. Therefore an RoI Pooling layer is introduced, which converts the provided proposed regions into feature vectors of fixed size. This improvement already led to a considerable time reduction of the

4 SVM's are used to classify data in classes in a form, that there is maximum space between the classes [16].

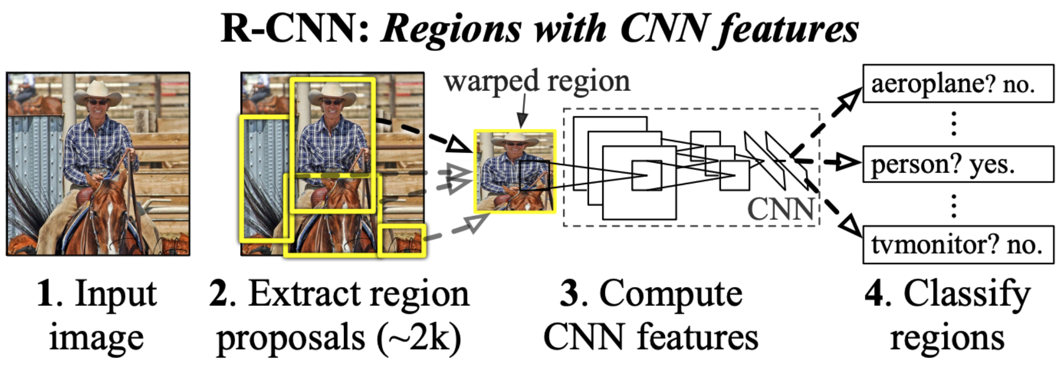


Figure 2.13: Architecture of the R-CNN network [21]

training and also the testing. It also showed that the region proposals now dominate the remaining time it takes to train and test the model.
This problem was then tackled in the "Faster R-CNN" paper [56]. To get rid of the slow region proposals provided by Selective Search, a region proposal network was introduced. With the help of this network and a few other tricks the training and also evaluation times could be improved.

2.3.2.6 *Mask R-CNN*

Mask R-CNN is a further improvement of the Faster R-CNN. It adds a third output, which is a per-pixel basis mask of the found object. This is done based on the Faster R-CNN network by adding a third branch. This third branch consists of a fully connected network which is applied to each RoI in order to predict a segmentation mask. In order to get a precise enough alignment to make this per-pixel mask branch, mask R-CNN improves the RoI pooling layer from Fast-RCNN. In the paper, they call this improved version of the RoI pooling layer the RoIAlign layer. The RoIAlign layer intends to fix the location misalignment, which is caused by quantization in the RoI pooling layer. An example output of Mask R-CNN is shown in Figure 2.14. It is impressive on how precise the segmentation is, but it also shows some weaknesses, for example, in the bottom right, where the chair contains some pixels of the floor.

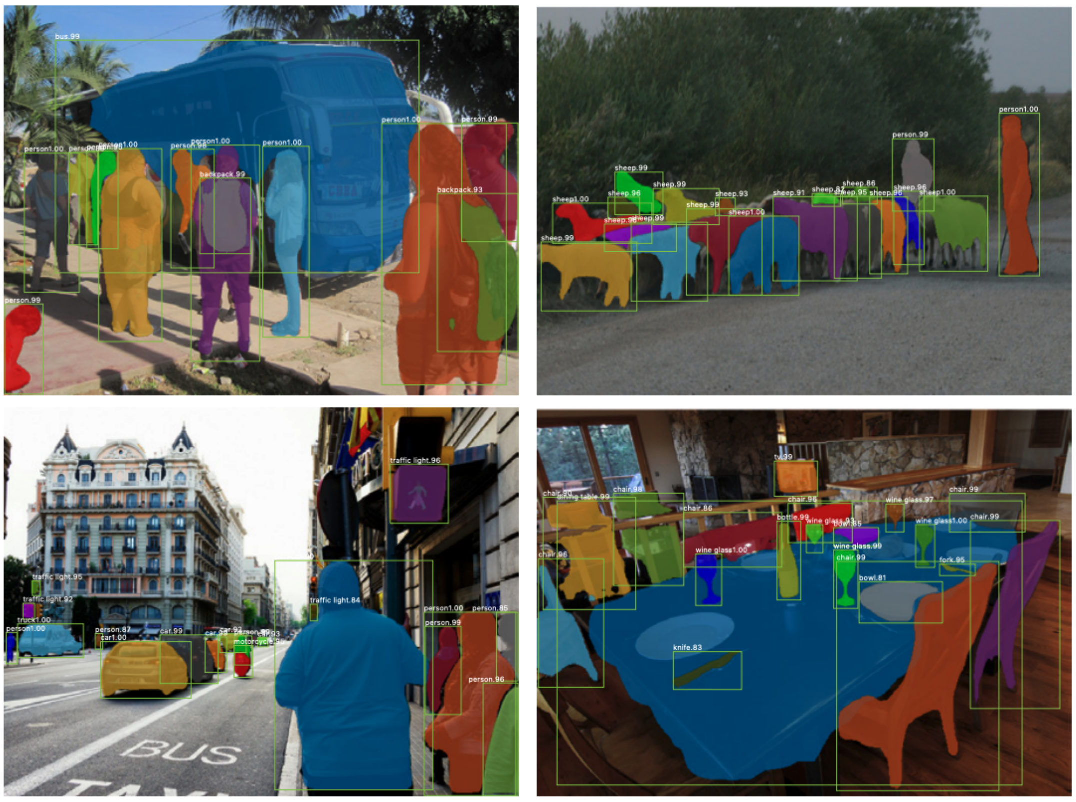


Figure 2.14: Example images of Mask R-CNN. Each found occurrence is marked by a bounding box, the mask pixels and a class with a confidence score.

2.3.2.7 *Summary*

In this section, many different architectures have been shown. With AlexNet showing off the capabilities of Convolutional Neural Networks, a hype was started, and lots of time and money was spent on improving these networks. Deeper networks perform better, but that deeper networks also result in more parameters and thus more computational power is needed. To be able to still handle deeper depths of networks inception blocks and stacking of smaller convolutions were introduced. With the introduction of ResNet and its residual connections, even deeper networks were possible and showed that they achieved the point where the network achieved better results than an actual human being [60]. Furthermore, with Mask R-CNN and its previous papers, it was made possible to get not only classification but also an instance segmentation of the input image. The results of Mask R-CNN look promising and hence, will be used in order to get a segmentation and classification of food images in the project.

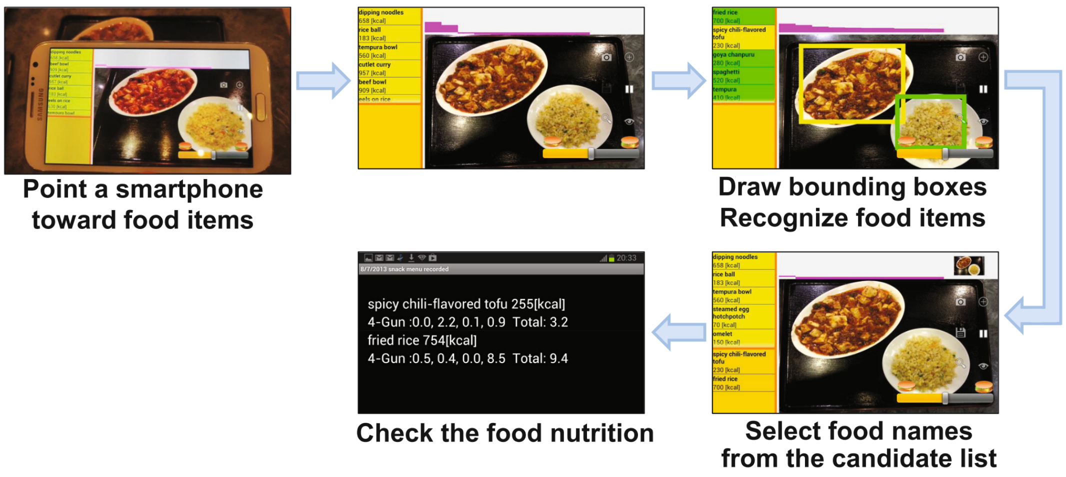
3 RELATED WORK

Food image recognition and detection is not a new topic. Much effort has already been put in to create a system that can detect food images reliably. As food images are particularly challenging, because even food images showing the same class can vary a lot in apperences, different approaches have been proposed, and the most popular ones are presented in this chapter.

3.1 FOODCAM

FoodCam [38] was one of the first papers which got a mobile version of food classification and calorie estimation to work. The contribution of this paper was to show that a mobile version of such a detection system was already possible. This should help the user to track their food intake more easily. To make this work, it relied on handcrafted features optimized to run fast on a smartphone. To get a classification estimate, the user has to point the camera of the smartphone towards the food items. Next, a bounding box has to be drawn around the food items in the image. The features are then extracted only from the parts which contain food. After the application has classified the image regions, it proposes a candidate list to the user. The user needs to click on the correct candidate and input the food item's rough volume with the help of a slider. The result is then the selected class and the nutrition value of that food (see Figure 3.1). In the paper, they did not state how this nutrition value is calculated. Their approach could score an accuracy of about 79% on the UEC dataset for the top-5 classification accuracy (subsection 4.2.2). The UEC dataset is a dataset of mainly Asian dishes gathered at the University of Electro-Communications.
In 2016, Tanno et al. [68] improved this approach by using a deep convolutional Neural Network instead of the handcrafted features. They named this new version DeepFoodCam. With this improved version, they could increase the accuracy up to a value of 95% while keeping the processing time at a similar level [68].
The application can be downloaded from their website[1]. One problem with their approach is that they trained the network mainly on a dataset containing Asian food items. This leads to worse results when testing the app in a European context.

1 Foodcam, http://foodcam.mobi/system.html, accessed: 02.07.2020


Point a smartphone toward food items
Draw bounding boxes Recognize food items
Select food names from the candidate list
Check the food nutrition

```
spicy chili-flavored tofu 255[kcal]
4-Gun :0.0, 2.2, 0.1, 0.9  Total: 3.2
fried rice 754[kcal]
4-Gun :0.5, 0.4, 0.0, 8.5  Total: 9.4
```

Figure 3.1: Possible workflow of the FoodCam application. [38]

3.2 IM2CALORIES

In 2015, Myers et al. published the Im2Calories paper [52]. In this paper, they proposed a food recognition and calorie estimation system. Their system makes use of two approaches. The first one is to find out if a person is eating in a restaurant, of which the app knows the menu, and, therefore, can look up the calorie amount on the restaurant's website. Moreover, the second is to predict the category and calorie amount of non-restaurant images. A graphic of the complete process, which is performed to get a classified, segmented, and estimated result of a food image, is shown in Figure 3.2. To achieve the first, they trained a GoogLeNet CNN model [67] to detect if there is food present in the image. Then they try to find the current user's restaurant via Google's Places API[2]. Once the restaurant is known, a web search is performed to receive pictures of that restaurant's menu items. After that, a multilabel classification is executed in order to get the best matching entry. With this entry known, one can simply look up the calorie values of that specific kind of menu in a food database. This approach only works for bigger restaurants, which have many pictures of their food and the respective calories published on the web. Due to a law from 2018, which forces restaurant chains to disclose their meals' calorie and nutrition amount, these data are easily accessible, at least for larger restaurants in the United States. This approach makes the detection and calorie estimation much more reliable, as the meals are typically always prepared in the same manner. Nevertheless, this approach only works for restaurants, which have published food images and the corresponding calorie amount. As in general there is still a huge number of smaller restaurants which have not published their meals, this approach is still and probably will always be infeasible.
That is why they also developed a second approach which works for generic food image detection. For this, they took a part of the Food101 dataset and built two further datasets, which they named Food201-MultiLabel and Food201-Segmented. The two datasets will be presented in more detail in subsection 4.2.4. It is sufficient to

2 Google's Places API, https://cloud.google.com/maps-platform/places/, accessed: 25.06.2020

know that the multilabel dataset adds multiple labels per image, and the segmented dataset adds ground truth segments to the dataset. These two new datasets make it possible to classify one label per image and detect multiple items in the image. Furthermore, it can tell the class and predict the segments that belong to the different classes. With the segments known, volume estimation can be performed. They trained another CNN network based on Eigen et al. [14], which predicts the depth for every pixel from the camera image. Next, they convert this depth map into a voxel representation. To get the food's calorie amount, the algorithm looks up a calorific density value of the detected kind of food on one of the food databases. In the paper, they state that it is best to use the USDA Food and Nutrient Database for Dietary for cooked meals[3]. Nevertheless, they also mention that it is hard to predict the calories correctly, even if the volume and the meal class are known. This is because even the same meals can be cooked with different ingredients, which makes it hard to detect the correct calorie amount reliably.

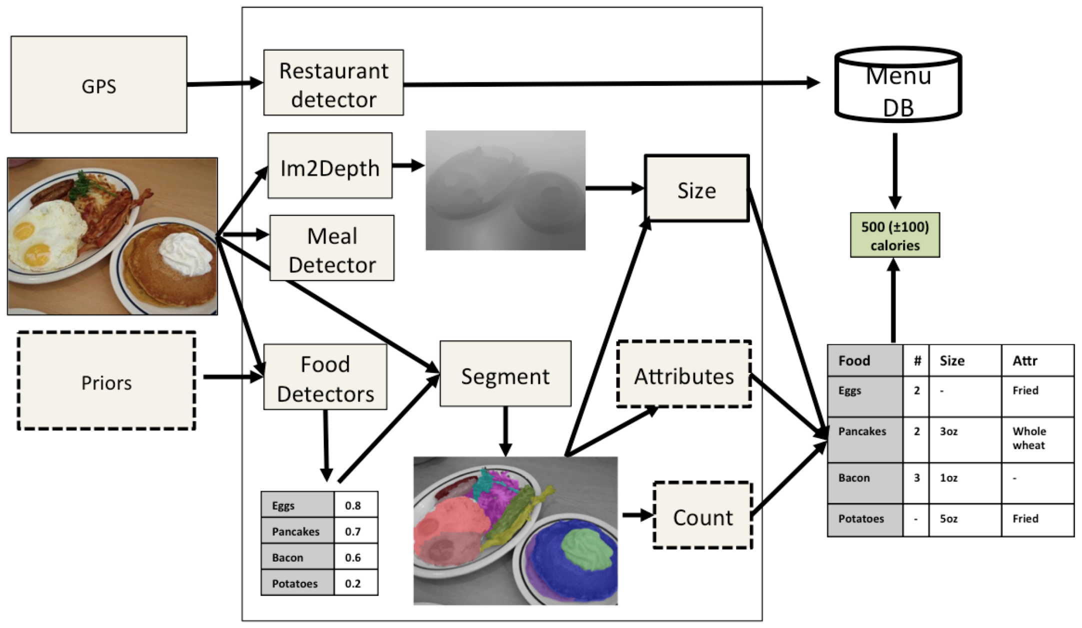


Figure 3.2: Overview of the Im2Calories system [52]. Dashed lines are parts, which are future work and not implemented in the paper.

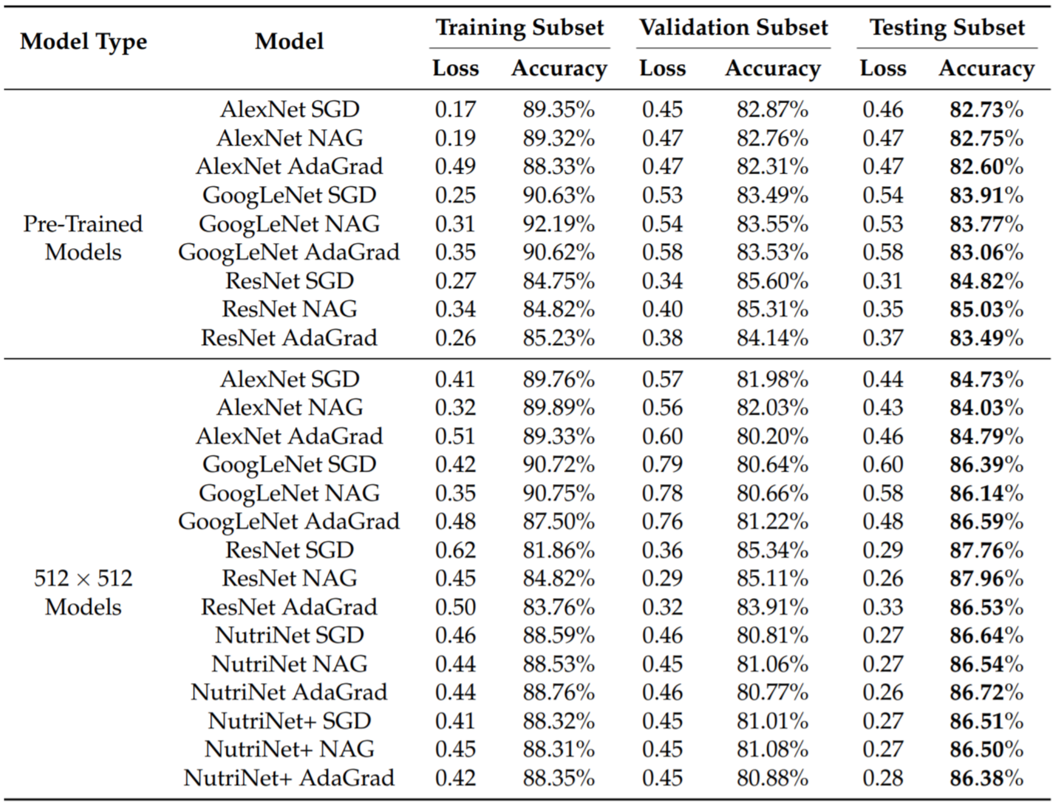
3.3 NUTRINET

With NutriNet [51], Mezgec and Seljak had a practical approach in mind. Their goal was to use the developed system in a mobile application for the dietary assessment of Parkinson's disease patients. The paper's main contribution is two-fold: First they present a newly defined network called NutriNet, a deep convolutional Neural Network architecture, and second, they created a food and drink image recognition dataset, which contains 225.953 images for 520 different classes of food and drink items. Although they have not released the dataset itself, they have released a set of

3 USDA Food and Nutrient Database, https://fdc.nal.usda.gov/, accessed: 25.06.2020

tools that they have used in order to generate that dataset. This should help other researchers replicate the dataset or generate a custom one for their personal use case. The problem with this tool-set is that they used the Google Custom Search API[4] in order to generate the dataset and that Google has changed their terms of use in such a way.
The authors also provided a detailed description of how they cleaned the crawled data. They trained another Neural Network to distinguish food from none food items and only keep the crawled pictures containing food items. This approach is especially helpful when implementing a network, which should learn new classes provided by the users. A general use case would then be as follows: A user uploads a picture and recognizes that the network's provided class is not correct. The user will then provide the correct label. To retrain the new class network, an image search with the help of the Google Custom Search API would be performed to gather more images of that particular class. The food detection network would then try to find images containing food and discard the others. Once this is done, the actual network can be retrained with the complete original dataset and the new class's newly gathered images.
The result is a new model, which will correctly classify the provided image into the new class. This procedure hopes that the network improves over time and that the users get a more precise classification. In the NutriNet paper, the authors implemented such an approach but did not further evaluate this learning step. Nevertheless, they did compare their new network to the other common network architectures, and it turned out that their network outperformed AlexNet and performed similarly to the GoogLeNet architecture. Only the ResNet architecture performed better. In the paper, they state that ResNet could achieve better accuracy due to the network's higher depth, but that this depth also increases training time. Moreover, as they retrain the network frequently, this training time would be too high. The comparison of the different architectures can be seen in Figure 3.3. They also compared different optimizers, which will not be discussed further. They used their app in a dietary assessment of Parkinson's disease patients and therefore developed a mobile application. With the app's help, the participants gathered real-life images of their meals, which could then evaluate their system on real-life images. On those images, the NutriNet network could achieve a top-5 accuracy of 55%. They state that this low accuracy comes from the fact that these real-life images contain multiple food items, and the network can only predict one class. This limitation is not tackled in their paper and is left for future work.

4 Google Custom Search API, https://developers.google.com/custom-search, accessed: 18.07.2020



Model Type	Model	Training Subset		Validation Subset		Testing Subset	
		Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
Pre-Trained Models	AlexNet SGD	0.17	89.35%	0.45	82.87%	0.46	**82.73%**
	AlexNet NAG	0.19	89.32%	0.47	82.76%	0.47	**82.75%**
	AlexNet AdaGrad	0.49	88.33%	0.47	82.31%	0.47	**82.60%**
	GoogLeNet SGD	0.25	90.63%	0.53	83.49%	0.54	**83.91%**
	GoogLeNet NAG	0.31	92.19%	0.54	83.55%	0.53	**83.77%**
	GoogLeNet AdaGrad	0.35	90.62%	0.58	83.53%	0.58	**83.06%**
	ResNet SGD	0.27	84.75%	0.34	85.60%	0.31	**84.82%**
	ResNet NAG	0.34	84.82%	0.40	85.31%	0.35	**85.03%**
	ResNet AdaGrad	0.26	85.23%	0.38	84.14%	0.37	**83.49%**
512 × 512 Models	AlexNet SGD	0.41	89.76%	0.57	81.98%	0.44	**84.73%**
	AlexNet NAG	0.32	89.89%	0.56	82.03%	0.43	**84.03%**
	AlexNet AdaGrad	0.51	89.33%	0.60	80.20%	0.46	**84.79%**
	GoogLeNet SGD	0.42	90.72%	0.79	80.64%	0.60	**86.39%**
	GoogLeNet NAG	0.35	90.75%	0.78	80.66%	0.58	**86.14%**
	GoogLeNet AdaGrad	0.48	87.50%	0.76	81.22%	0.48	**86.59%**
	ResNet SGD	0.62	81.86%	0.36	85.34%	0.29	**87.76%**
	ResNet NAG	0.45	84.82%	0.29	85.11%	0.26	**87.96%**
	ResNet AdaGrad	0.50	83.76%	0.32	83.91%	0.33	**86.53%**
	NutriNet SGD	0.46	88.59%	0.46	80.81%	0.27	**86.64%**
	NutriNet NAG	0.44	88.53%	0.45	81.06%	0.27	**86.54%**
	NutriNet AdaGrad	0.44	88.76%	0.46	80.77%	0.26	**86.72%**
	NutriNet+ SGD	0.41	88.32%	0.45	81.01%	0.27	**86.51%**
	NutriNet+ NAG	0.45	88.31%	0.45	81.08%	0.27	**86.50%**
	NutriNet+ AdaGrad	0.42	88.35%	0.45	80.88%	0.28	**86.38%**

Figure 3.3: Performance comparison of different network architectures on the custom dataset generated in the paper. Image taken from NutriNet paper [51]

3.4 WIDE-SLICE RESIDUAL NETWORKS FOR FOOD RECOGNITION

The previously presented works have one thing in common. They all use existing off-the-shelf deep architectures to classify food items [49]. That means that the architectures were not explicitly designed for the context of food images. Martinel et al., propose with the Wide-Slice Residual Networks for Food Recognition (WISeR) a new approach that considers the structure of food images. To achieve this, they developed a slice convolution block in order to capture the vertical food layers. In contrast, a standard convolution layer commonly only considers a square kernel, the slice convolution block aims to capture the food images' vertical layer structure. Due to the fact, that these vertical features might be present in some but maybe not in all images, they fuse a standard residual network with this slice convolutional branch. Due to this improvement, the authors could achieve remarkably high accuracy. They achieved an accuracy of 90.27% top-1 and 98.71% top-5 on the Food101-dataset. Five sample images can be seen in Figure 3.4. The green bars indicate the correct label, while the red bars indicate wrong labels. The network only performs classification, but no segmentation of the input image and is performed on the whole image and not on the different instances of the food items. Another limitation they state in their paper is that the proposed solution requires a substantial memory load and significant computational efforts, denying the possible deployment of a mobile phone approach.

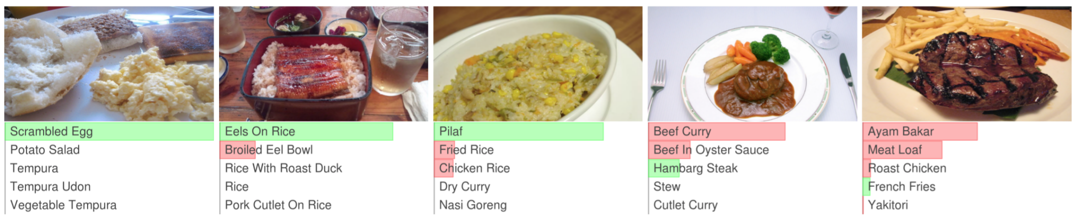


Figure 3.4: Top-5 predictions of the Wide-Slide Residual Networks for Food Recognition architecture on images taken from the UECFood256 dataset. Green bars indicate the correct prediction, red bars indicate incorrect matches [49].

3.5 CNN-BASED FOOD IMAGE SEGMENTATION WITHOUT PIXEL-WISE ANNOTATION

Shimoda et al. propose a method that enables classification and segmentation of food images without requiring pixel-wise ground truth data [64]. Therefore, an algorithm was developed, which performs the steps shown in Figure 3.5. The approach is a fusion of R-CNN [21] and the paper of Simonyan et al. [65]. It takes the regional approach from R-CNN and combined that with the salience activation maps from Simonyan et al. To get the regions similar to R-CNN, Selective Search is performed in a first step. As a result of the Selective Search, they receive 2000 bounding boxes, which probably contain food items. To not process these boxes individually, the algorithm performs merging operations and thus reduces the number. For every candidate, a Backpropagation (BP) is evaluated to generate a salience map of the input image. Here the idea is that the pixels that correspond to the actual food image are the most active pixels in the BP step. The details of this algorithm exceed the scope of this work, but are explained in detail in their paper [65]. Because the BP step only estimates the most discriminative parts of the object GrabCut [58] is performed to get the actual segments of the food parts. At last, a non-maxima suppression is calculated to avoid multiple occurrences of the same segment. These multiples could arise due to the candidate creation, which allows overlapping segments. This here presented approach is interesting, as it does not require actual ground truth pixel data to produce a segmentation output, and it also takes multiple occurrences in the food image into consideration and not only tries to classify the complete image. The authors state in the paper that they could outperform R-CNN by far.

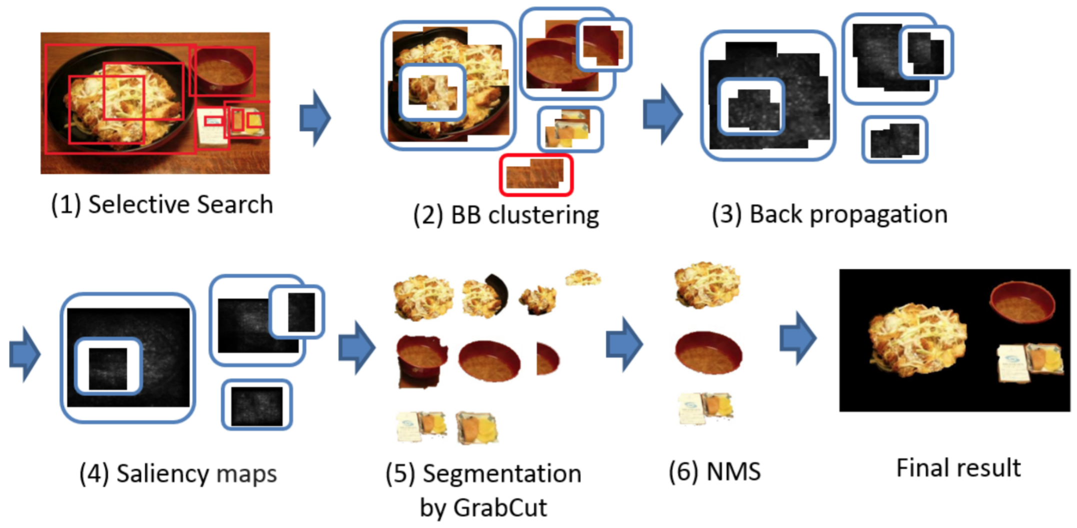


Figure 3.5: Workflow of the CNN-based food image segmentation method [64].

3.6 DEEP LEARNING-BASED FOOD CALORIE ESTIMATION METHOD IN DIETARY ASSESSMENT

Liang and Li chose a similar approach to the before-mentioned approach by Shimoda et al. [64]. However, instead of focusing on object detection, they focused on the volume- and calorie estimation. This was possible since Faster-R-CNN already achieves a proper object detection. The output of Faster-R-CNN is a set of bounding boxes and classes of the found food items. To receive not only the bounding boxes but also a pixel-wise segmentation, they use GrabCut [58] on the boxes. GrabCut is an algorithm that tries to separate the background of an image from the corresponding foreground. This helps to have only parts in the segments containing food and therefore reduce the noise in the segments. To improve the later volume estimation, they require the user to take two pictures of the food. One from a side view and one from a top view (see Figure 3.6). In both of these images, the user needs to place a reference object. In their paper, they used a one Yuan coin. After detecting the different food items in both images, the reference object can be used as an anchor to calculate the volume with the two perspectives' help. Once the volume is calculated, the food's density value needs to be looked up in some food database and can then the correct calorie amount can be calculated. To evaluate their system, they created their dataset, which mostly consists of fruits. Thus, the setup is experimental and would probably work worse in a real-life setting. The fruits are placed on a white plate on their own. This makes it easier to detect them then on a real-life picture where different noise is present in the images. Furthermore, they match the items of the two images based on the class of the found objects. They do not state if this would still work if multiple instances of the same class would be present in one image.

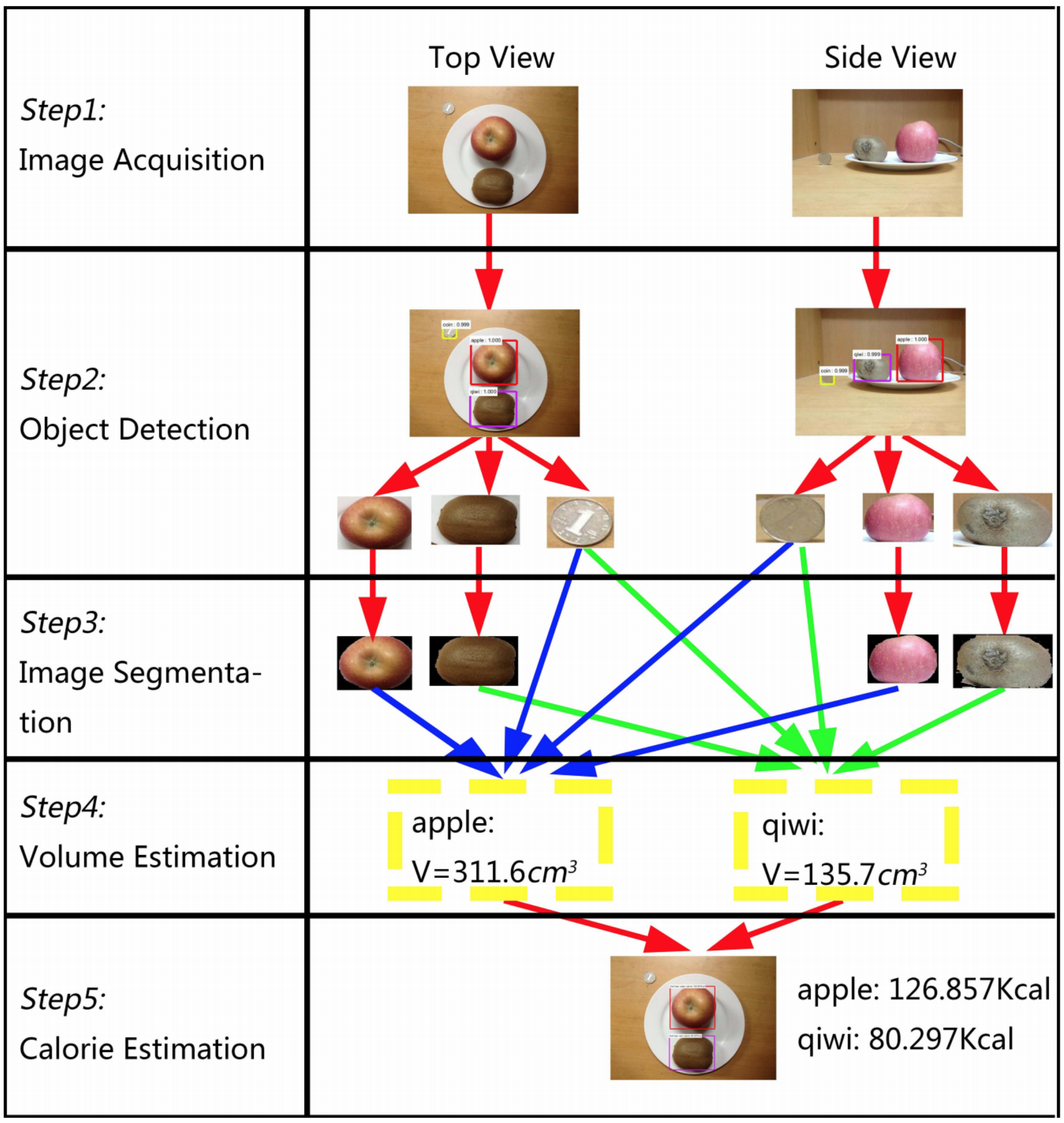


Figure 3.6: Flowchart of the proposed system by Liang and Li [44].

3.7 SINGLE-VIEW FOOD PORTION ESTIMATION

Although often used, convolutional Neural Networks are not the only method to get an accurate calorie estimation. Another family of networks was used in the paper from Fang et al. [18], the Generative Adversarial Networks (GANs) [22]. The theory behind GANs is a two-player game on which one player generates new images, and the other player needs to detect if the generated image is real or fake. In recent years GANs have gained much attention as they have been proven to be very efficient in generating fake images [63]. There is no need to perform the often used pipeline of object detection, segmentation, volume estimation, and calorie estimation with this approach. In this specific case, the network will directly output a per-pixel energy value. This energy value should represent the calorie amount on that specific pixel. An example of such a mapping is shown in Figure 3.7. To achieve such a result, the network needs a dataset that provides all the relevant information. As no publicly available dataset offered such information, they created a new dataset and, manually labeled 202 training food images and 220 testing food images. Fang et al. report an energy estimation error of 10.89%. This is better than previous works which used a geometric model-based technique that relied on accurate food segmentation and classification [18]. Of course, this is hard to compare, as they did not evaluate other methods on the same dataset.

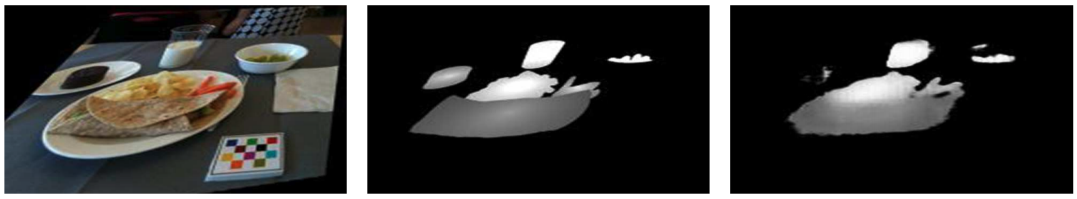

(a) Eating occasion image $\bar{\mathbf{I}}$. (b) Ground truth energy distribution image $\bar{\mathcal{W}}$. (c) Estimated energy distribution image $\tilde{\mathcal{W}}$.

Figure 3.7: A sample image of the image to energy mapping [18]. (a) The original rectified image. (b) The ground truth energy distribution. (c) Output of the trained network.

3.8 SUMMARY

Foodcam was the first application, which aimed to get a mobile version of digital food analysis to work. With this application, Kawano et al. [38] proofed that it is possible to run such a task on a mobile device. However, this application's drawback is that it is mostly trained on Asian food images and therefore not usable for food of other regions.
Im2Calories provided a two-folded approach. The first one is to use the restaurant menu, while the second is to detect images with an unknown menu. The second one is more promising, as it does not rely on data provided by the restaurants. The authors only converted the classification part of the detection pipeline to a standalone mobile application. The other parts could still be used in a server-client architecture. At the end of their paper the authors claim, that calorie estimation is hard, as food parts could be occluded or the meal could be prepared with different ingredients.
NutriNet created and published a framework to help other researches to create their custom datasets. Therefore, they trained a network that decides which of the gathered images contain food and thus should be used. In their paper they tested the application on real-life images and showed that the accuracy is with 55% worse than on the crawled images (86%). This worse accuracy is because real-life images are often more complex and contain more noise than crawled images. The process to crawl the internet for images of previously unknown classes is an exciting idea and could help to improve networks continuously. As this change in the accuracy would disturb a study setting, it is not used in this work but could be further investigated in future work.
WISeR tried to use a custom convolution structure to make use of the structure of food images. Due to that approach, the authors could achieve remarkably high accuracy of 98.71%. This new approach requires a substantial memory load and therefore denies a mobile phone approach. This limitation could be bypassed with a server-client architecture, but a server-client architecture has other limitations like

the need for an active network connection.
Shimoda et al. proposed a method that does not need a pixel-wise annotation and enables instance segmentation. Although this method outperformed R-CNN, the newer version of R-CNN, called Mask R-CNN outperforms the work of Shimoda et al. Therefore, Mask R-CNN is preferred as basis.
Liang and Li used Faster R-CNN as a detection network and focused on the food items' volume estimation. Therefore, they require the user to take two pictures, one from a side-view and one from a top-view. With those images and a reference object of known size, they can calculate a volume estimate and then look up the specific kind of food's calorie amount. Although this approach was only shown in an experimental setup, having a reference object of known size in the image is a good idea and should also be used in the future project.
Fang et al. used a different approach by using GANs to directly generate an energy map of the input image instead of convolutional Neural Networks. GANs is an exciting idea, but are beyond the scope of this work and could be further investigated in future work. For this work, this means that restaurant data is not useful, as the more common use case is that the users will take a photo of self-cooked dishes. Furthermore, the task of achieving an accurate calorie estimation seems to be very challenging. This comes from the fact that occlusion of food parts and a high intraclass variance for the different food classes makes an accurate prediction almost impossible.

4 DEEP LEARNING IN CONTEXT OF FOOD IMAGES

Machine learning heavily depends on the given input data. Therefore in this chapter, the publicly available datasets will be presented. Due to the lack of a food image dataset with millions of images, it is common practice to use a model already trained on other non-food images. This is reasonable as the general features (for example edges) learned in earlier layers of a convolutional Neural Network might also work in food images. Transfer learning is then used afterward to fine-tune the network to the domain of food images. Therefore, in the first part of this chapter, the two most famous publicly available datasets for general images will be presented. After that, the available food datasets will be presented. The next section presents the requirements this work has on a dataset and compares the different existing datasets.

4.1 LARGE GENERIC IMAGE DATASETS

In recent years more and more image datasets were published (e.g ImageNet [15], PASCAL Visual Object Classes dataset [43], Open Images Dataset V4 [13] and Mircosoft common objects in context dataset [46]). Additionally, existing datasets are continuously improved. Because the available data is crucial for the later training of the network, large companies like Microsoft, Google, and Amazon invest much money to help improve these large datasets.

4.1.1 *ImageNet*

The developers of ImageNet describe it on their website as "[...] is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet, majority of them are nouns (80,000+). In ImageNet, we aim to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, we hope ImageNet will offer tens of millions of cleanly sorted images for most of the concepts in the WordNet hierarchy." [33]

ImageNet has already gathered over 14 million images (as of 23. May 2020). The database consists of various image categories. All of the available images have ground-truth classification labels, and for many images, ground truth bounding boxes are provided. To achieve such a huge amount of images, they crawled the web with different image search engines and different languages and synonyms of the actual word. To clean the data, they used a service called Amazon Mechanical Turk[1], a crowdsourced service hosted by Amazon. With this service, one can pub-

1 Amazon's Mechanical Turk,https://www.mturk.com/, accessed: 25.06.2020

lish jobs and pay workers. A more detailed explanation of how they gathered and polished the images is stated in their paper [13]. ImageNet became very popular, as they hosted the yearly ILSVRC-challenge. For this challenge, the most popular architectures for convolutional Neural Networks were developed. These architectures were already presented in section 2.3.

4.1.2 *Microsoft COCO*

The Microsoft Common Objects in Context (COCO) dataset is with 330.000 images smaller than the ImageNet dataset. Thus, the COCO dataset has the advantage that it provides an instance segmentation of all of the images. Therefore, the use case of the COCO dataset is different than for ImageNet. While ImageNet is mostly used for classification tasks, the COCO dataset is mainly used for segmentation tasks. To decide which classes they should support in the COCO dataset, they combined the categories from PASCAL VOC [15] and a subset of the 1200 most frequently used words that denote visually identifiable objects [46]. To further augment the set of class candidates, they asked children from the age of 4 to 8 to name every object they see in indoor and outdoor environments. This resulted in 272 candidates, which were then further filtered by the paper's co-authors and thus reduced to a final amount of 91 categories. Then they crawled images for each of these categories from an image hosting website. To receive images that contain not only the actual categories but also additional elements, they combined different categories in the search term and, for example, searched additionally to "car" also for "car + dog". To get the ground truth data for the collected images, they also hosted tasks on Amazon's Mechanical Turk.

4.2 FOOD-IMAGE DATASETS

The next sections will provide an overview of the available food datasets.

4.2.1 *Pittsburgh Fast-Food Image Dataset (PFID)*

With the PFID dataset, the first publicly available food dataset was released in 2009. It contains 101 different foods from 11 different fast-food restaurants for a total of 4.545 images. They gathered pictures of real-life settings as well as from a lab setting (see Figure 4.1). The reason for why the authors chose fast food pictures as the domain is that the gathering of ground truth data is straightforward. The restaurant provides the class for every meal, and every restaurant should prepare the meal very similarly. So the intra-class variance should be low. Besides, fast-food restaurants often provide a list of ingredients and the number of calories of the different meals. Furthermore, the authors claim that fast foods are the key to obesity research [6]. One of the problems with that dataset is that the images are only taken from 11 fast-food restaurants in the US. Although a few may also be present in other countries, this limits the use of this data to train a generic food detection system.

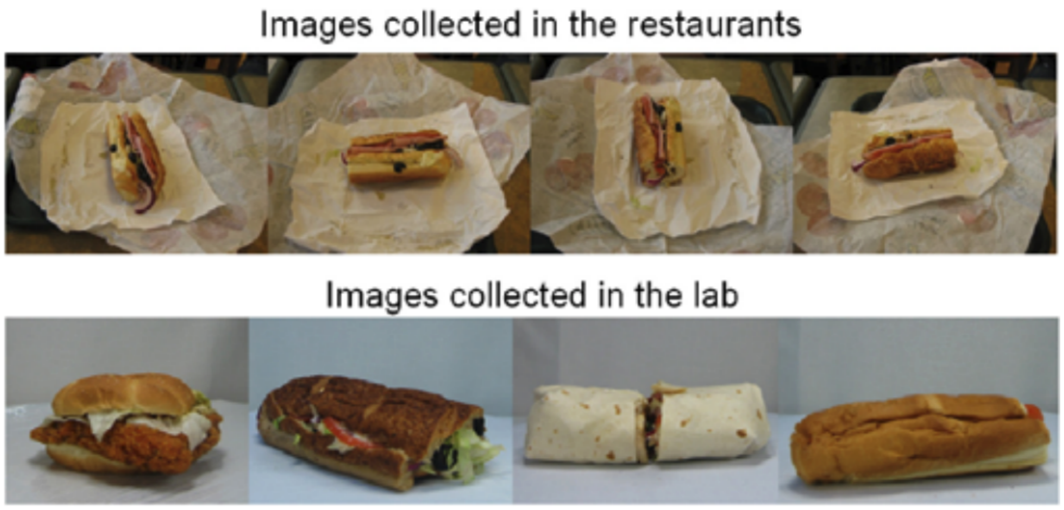

Images collected in the restaurants
Images collected in the lab

Figure 4.1: Sample images of the PFID dataset. Image is taken from [6].

4.2.2 *UEC-Food 100/256*

In 2012, Matsuda et al. released the UEC-100 dataset [50]. This dataset contained 100 different food classes (see Figure 4.2 for some examples). The problem is that the goal of Matsuda et al. was to create an efficient food recognition system that is intended to be used in Japan. Therefore, food classes are mainly traditional Japanese food.
Nevertheless, they provided 9060 images, that all have been labeled, classified, and provided with bounding boxes.
Two years later, Kawano et al. recognized this shortcoming of only having Japanese images and expanded the dataset [39]. They named the new dataset UEC-Food256 and added 156 classes of more international food items to the dataset. Due to that, the number of images was increased to a total amount of 31.397. The already in section 3.1 presented network was trained on that dataset and showed promising results. To gather the additional data, they crawled the web (Flickr API, Twitter API, and Bing Image Search API). To automate the process as much as possible, they trained a SVM on the original UEC-Food100 dataset to check if the crawled images contain food. With another SVM, they tried to find similar-looking images for the searched class and discard the not similar ones, as they probably contain the wrong items. Afterward, they used Amazon Mechanical Turk to crowdsource the task of generating bounding boxes for the collected food items.

Figure 4.2: Left: Images of the UEC-Food100 dataset. Right: Images of the UEC-Food256 dataset. Image taken from [49].

4.2.3 *Food-101*

Food-101 is the largest food dataset that has been gathered so far and is publicly available. It contains 101 classes and for each class 1000 images. This results in a total amount of 101.000 images. From the 1000 images, there are 250 testing images, which were manually cleaned. The other 750 training images were not cleaned on purpose. In the paper, they state it is a requirement that real-world computer vision algorithms should be able to cope with such weakly labeled data [5]. The noise comes in form of intense colors and sometimes wrong labels. To gather this vast amount of data, they crawled foodspotting.com, a website where users could upload their meal pictures and label it. Sadly, foodspotting.com was taken offline in May 2018. Nevertheless, the Food-101 dataset grew to the most crucial dataset as it enables researchers to train and compare their networks on a large publicly available dataset. One limitation with the dataset is that it only provides the class for the specific image. This also means that there is only one item at a time on the image. Therefore, this dataset cannot be used if the goal is, for example, to train the network to do segmentation.

4.2.4 *Food-201*

The problem, that Food-101 only provides one class label as ground truth was the reason for Myers et al. to enhanced the Food-101 dataset. They expanded the dataset to contain an additional 100 classes and provided two different versions of this new dataset. The first version is called Food-201-Multilabel, and the second one is called Food-201-Segmented. To generate the Food-201-Multilabel dataset, they took 50.000 images from the Food-101 dataset and created an Amazon Mechanical Turk task to label these images. They included the original class label as one of the choices, and manually created a list of commonly co-occurring foods as additional choices. Furthermore, they allowed the workers to enter new terms in a text box. After manually merging synonymous terms, and removing terms that occurred less than 100 times, they ended up with a vocabulary of 201 labels [52]. To generate the second dataset, they took 12.000 images from the Food-101 dataset and asked raters on a crowd computing platform to manually segment each of the classes that have been tagged (see Figure 4.3 for some examples). With this new dataset, it was possible to train a network that performed classification and segmentation. The problem with this dataset is that they have labeled the image on a per-class basis and not on an instance basis. This means that if two slices of bread are present in the image, then the ground truth only shows one label for both and not one for each of them.

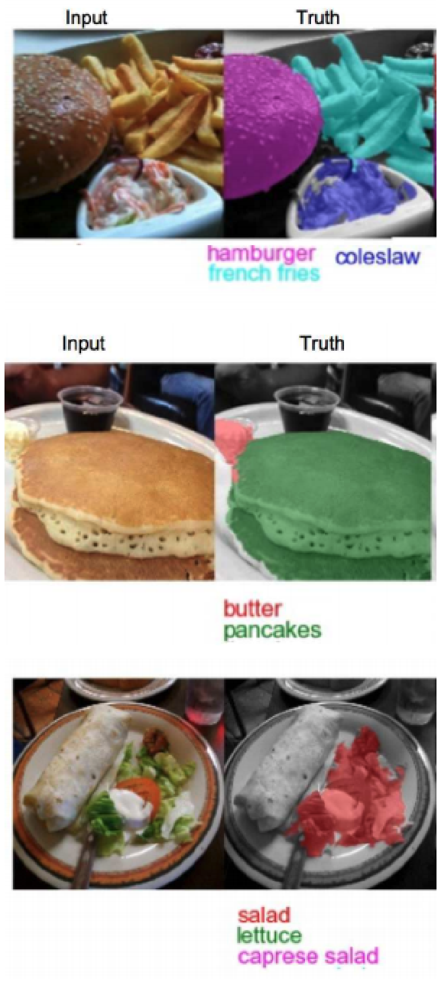


Figure 4.3: Sample images from the Food201-Segmented dataset. Image taken from [52].

4.2.5 *UNIMIB*

In 2015 another food dataset was released, the UNIMIB2015 dataset by Ciocca et al. [9]. This dataset contains canteen tray images. Each of these tray images contains multiple food items and some noise, for example, a cellphone or wallet. The dataset was created to enable food recognition and leftover estimation. For this reason, they have two images for each meal. One is taken with the full tray, and the other is taken when the tray is empty, or at least only contains the leftover of the meals. The dataset has only 15 classes, and the trays used were blue and, therefore, in a different color than the plates. This makes it easy to distinguish between the tray

and the plates.
To improve the dataset and make it more challenging, they created the UNIMIB2016 dataset one year later. This dataset consists of 1027 images for 73 classes. Additionally, they got rid of the blue trays and replaced them with white ones. It is more challenging to distinguish between the tray and the plates (see Figure 4.4, for examples). The authors manually labeled and segmented the images for both datasets (UNIMIB2015 and UNIMIB2016) and, therefore, provided ground truth polygon values for every image. Due to privacy issues and intense affluence of customers, they could not take pictures of the trays after the meal. Therefore, the newer dataset cannot be used for leftover estimation.

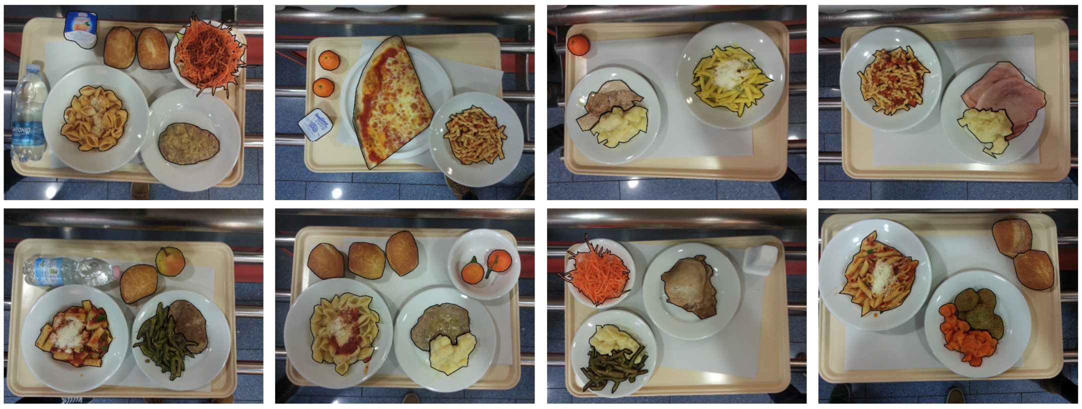


Figure 4.4: Sample images from the UNIMIB2016 dataset.The black polygone around the food represents the manual annotations. Image taken from [9]

4.3 REQUIREMENTS

To decide on an appropriate dataset, the following requirements have been gathered. These requirements arise from the planned work, which is to enable a food classification and segmentation for European foods, as well as from technical requirements of the chosen Mask R-CNN network architecture.

R 1. *The dataset needs to contain mostly European dishes.*

This requirement results from the fact that the app should be used in a European context, and hence, most of the dishes will be typical European food. However, this does not mean that the dataset should consist only of European dishes. The users could still eat international dishes, and therefore, the app should be able to handle those as well.

R 2. *The images need to be not staged food images.*

In the app, the users will take non-professional images of food. Therefore, it is important to train the network on real-life images and not on professional food images.

> **R 3.** *The dataset needs to provide ground truth segments and classes for each of the food items in the image. Moreover, there should be images with more than one food instance.*

Since the wanted output is an instance segmentation of the image; it is important that the training dataset also contains multiple labeled food items.

> **R 4.** *The dataset needs to provide a volume estimate for each food item.*

The network should be able to give a volume estimate for the individual detected food items. To enable that, it is mandatory to have the ground truth volume value of the training images.

> **R 5.** *There needs to be a predefined reference object in each image.*

To get a better volume estimation, it is important that an object of known size is in each of the images.

The outcome of the seminar paper [4] was a table with all the available datasets and their different attributes (see Table 4.1). The PFID dataset only fulfills R2, as the images consist of real-life fast food dishes, but only from North American restaurants. UEC Food-100 consists mostly of Asian food and only includes one item at a time with bounding boxes, but no segmentation is provided. UEC Food-256 tries to compensate for that and includes 156 international dishes. Still, the focus is on Asian images, and there are still no ground truth values for the segments. Food-101 greatly increases the number of available images, but still lacks segmentation ground truth data. Furthermore, it only contains one food item per image, which violates R3. Food-201-Multilabel improves this by adding additional food instances to the images, but still misses the ground truth segmentation values. Food-201-Segmented overcomes this limitation by adding ground truth values for the segmentation. Hence, Food-201-Segmented fulfills every requirement except for R4 and R5. UNIMIB2015 is an interesting dataset, as it consists of images from food trays of the canteen of the University of Milano-Bicocca. Therefore R1 - R3 are fulfilled, but still R4 and R5 are violated. Furthermore, R2 is only partially fulfilled, as the images are real-life images, but the trays and the plates are chosen in such a way, that it makes the detection easier (blue trays and white plates). UNIMIB2016 relaxes this constraint and uses normal colored trays, but still does not provide a consistent reference object, nor provides a volume estimate for the images.

This shows that no publicly available dataset fulfills all the requirements, and therefore in this work, a new dataset will be created. To achieve this, multiple steps need to be implemented. First, a massive amount of images needs to be collected. And second, for each of the gathered images, a ground truth value for every food instance needs to be created. These ground truth values must consist of a polygon containing the actual food parts, the related food class, and a volume estimate for each food item. A large number of images are available due to multiple studies run in the SMARTFOOD-Project [40, 41, 70, 71]. In those studies, different aspects of

Name	Year	#Images	#Classes	Type	Annotation	Fulfills
PFID [6]	2009	1098	61	Single	Label	R2
UEC Food-100 [50]	2012	9060	100	Single & Multi	BBox	R2
UEC Food-256 [38]	2014	31.397	256	Single & Multi	BBox	R2
Food-101 [5]	2014	101.000	101	Single	Label	R1, R2
Food-201-Multilabel [52]	2015	50.374	201	Single & Multi	Label	R1, R2, (R3)
Food-201-Segmented [52]	2015	12.000	201	Single & Multi	Poly	R1, R2, R3
UNIMIB2015 [8]	2015	1.000x2	15	Multi	Poly	R1, (R2), R3
UNIMIB2016 [9]	2016	1.027	73	Multi	Poly	R1, R2, R3

Table 4.1: Summary publicly available food datasets

eating behavior were investigated, but in all of them, they used an app developed in the SMARTFOOD-project. With the use of this app, the participants take pictures of their meals from a predefined angle (approximately 30 degrees) and were asked to place a one Euro coin next to their meal (see Figure 4.5). After taking the pictures, the participants described what they ate and gave a rough volume estimate of their meals.
This resulted in approximately 20,000, real-life food images. For each of these images, the participants have already named the different parts of the meals. Additionally, they have placed a reference object of known size next to their meal and have given a rough volume estimation of the meal's different parts. Hence, this dataset already fulfills almost all the requirements gathered at the beginning of this section. At this point, the class of the different items in the image is known, but the food parts' positions are still missing.

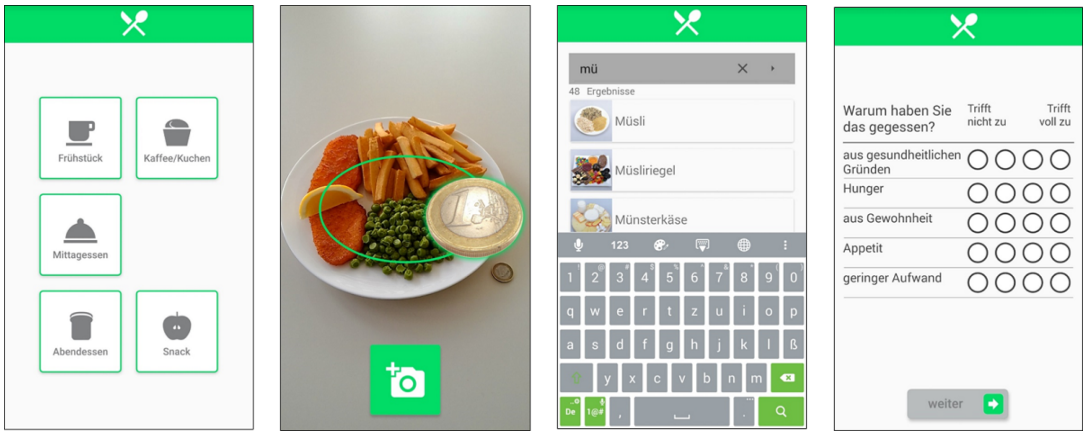


Figure 4.5: Workflow of the SMARTFOOD App to track food intake [71].

4.4 SUMMARY

When training data is sparse, it is common to train the network on a more general context and than only fine-tune the network on the actual context. The reason for this is that there are enormous datasets for more general contexts available. Furthermore, the hope is that the network will learn about features, which are also present in the intended context images.
Moreover, there are already various food datasets available. A summary of all the datasets is provided in Table 4.1. Each of the datasets presented has its advantages and disadvantages. Which dataset to use depends heavily on the use case. If, for example, a single classification is wanted, then the Food-101 is probably the best fit, as it provides many training images and 101 classes. If multi-classification is the goal, then UEC Food-256 or Food-201-Multilabel is a better fit. With its 156 classes more then UEC Food-100, it does not focus only on Asian food but adds some international dishes with multiple food instances per image. As it only provides 31397 images for more than two times the classes of Food-101, it maybe makes sense to combine the UEC Food-256 dataset with the Food-101 dataset. The same also applies to the Food-201-Multilabel dataset. It provides 201 classes and has around 50.000 images. If the goal is to perform a segmentation, the datasets get sparse. While Food-201-Segmented provides 12.000 images for 201 classes UNIMIB only provides 1.027 images for 73 classes. But UNIMIB has the advantage that it provides polygones for each food instance in the image, while Food-201-Segmented only provides one instance per food class. The UNIMIB dataset has the disadvantage of consisting only of canteen food images. These could limit the pratical use of this dataset in a real-life setting. Since non of the available food image datasets fullfills all requirements, a new dataset will be created based on the food images gathered during studies of the SMARTFOOD project.

5 IMPLEMENTATION SEGMENTATION

In this chapter the implementation of the segmentation pipeline is described. This pipeline is shown in Figure 5.1 and can be used to perform all the steps, which are necessary to go from unlabeled data to a trained network, which is then used in a mobile application to perform food image segmentation on new images. The pipeline consists of four step: (1) Generating ground truth values, (2) verifying and cleaning this data, (3) training the Neural Network and (4) deploying the trained model onto a server with which a mobile application can communicate. The sections in this chapter are organized according to these steps.

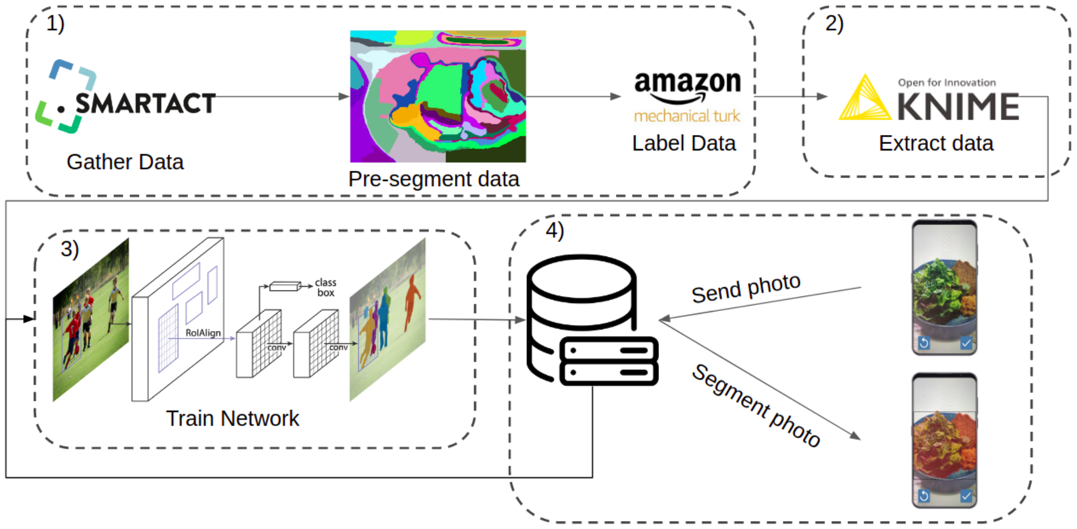


Figure 5.1: This figure shows a graph of the proposed pipeline. This pipeline is used to create a food detection model from given training data.

5.1 GENERATING GROUND TRUTH VALUES

The first step of training a Neural Network is to gather the images, that the network should be trained upon. These images were already collected in previous studies in the SMARTFOOD context. The images consist of information about the food categories on the images, but are missing the information where in the image these food items can be found (segmentation). Hence, in this section the approach of generating this data is described.

5.1.1 *Segmentation Interface*

To create the segments which contain food, a pre-segmentation algorithm was employed to divide the whole image into smaller parts. Afterwards, one has to click on the field, which contains food (see Figure 5.2 (b)). The idea was taken from the Bachelor Thesis of J. Wendler [72], which based her work on a Mean-Shift algorithm. This pre-segmentation should help speed up the process and make it easier to complete the segmentation task. In the work of Wendler, the Mean-Shift algorithm was run on a mobile device. Therefore, the available computational power was limited, and it was optimized to run in real-time on such a device and not optimized to get the best results. In this work, the pre-segmentation can be run on a desktop computer. Therefore, the algorithm's parameters were tweaked to get the best segments. Because this is a heavy computational operation, the algorithm was implemented so that it could be parallelized on all the available CPU cores. Additionally, a white balance was performed before the segmentation to counter lighting conditions in the images. After all these improvements, the algorithm still took two weeks to process all 20,000 images.

With the pre-segmentation completed, a web application was created, enabling users to click on segments and mark them as containing food. To enable this, each of the segments is scanned for its border pixels. Once these border pixels are found, only those pixels are displayed (see Figure 5.2 (c)). This way, it is still possible for users to see the underlying original food image. Furthermore, two different marking modes were implemented. With a left-click, the users indicate that the clicked segment solely contains food. A right-click indicates, that the segment contains food, but also some non-food parts (see Figure 5.2 (d)).

5.1.2 *Crowd Sourcing Services*

Many tools [31, 61, 74] have been proposed to make the generation of ground truth segments easier. However, creating ground truth values for 20,000 images still takes a considerable amount of time and attention. For that purpose, it is common to use crowdsourcing services to split the workload among different people. The concept of crowdsourcing is that a "requester" defines a job, which he makes publicly available. This job could be, for example, to segment 100 images. The so-called "workers" see the job on some platform and can decide to take it. If they complete the job and the requester is satisfied with the result, a previously defined amount of money is paid to the worker. Multiple platforms support such tasks. In this work, only the most prominent one will be presented, Amazon's Mechanical Turk (MTurk). This is because MTurk has a considerable worker-base and, more important, can be used for academic purposes.

5.1.3 *MTurk Templates vs. Custom Tasks*

For the creation of jobs on MTurk, there are two possibilities. The first one is to use a predefined template. Therefore, MTurk offers various templates for all kinds of tasks (see Figure 5.3). All of the templates come with an interactive preview, which helps determine if the template has all the required features. Once a template is chosen, one needs to specify some parameters. The parameters consist of the job's

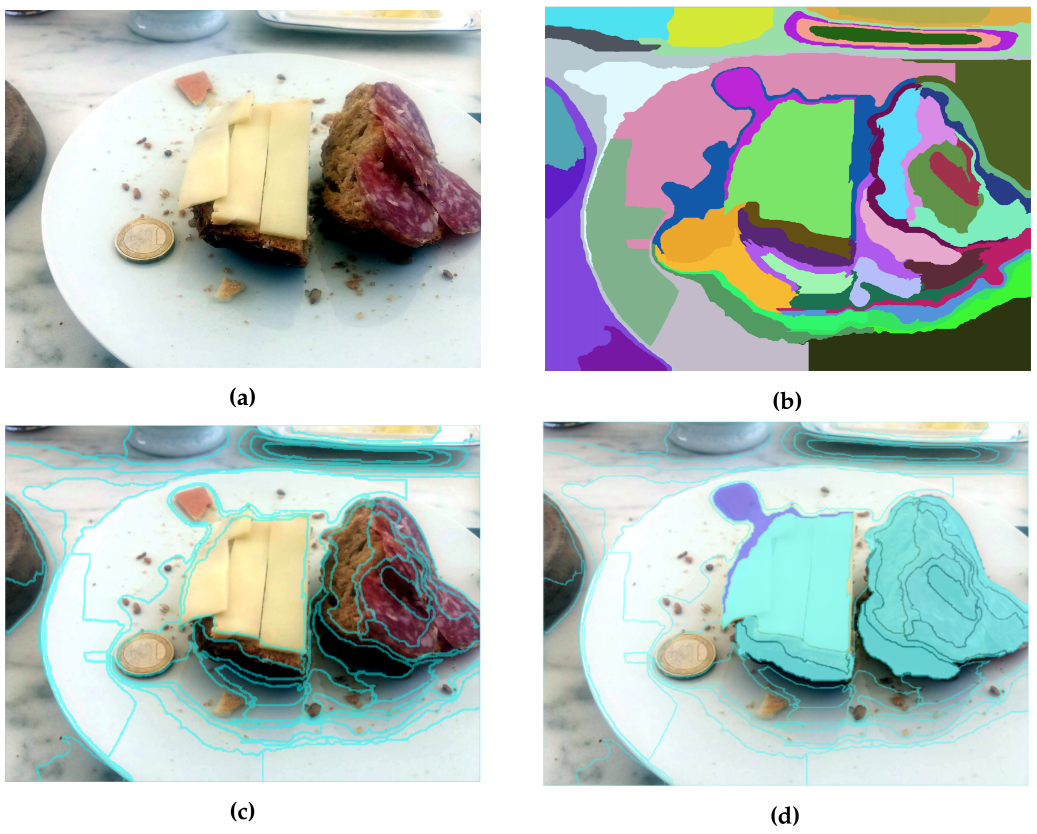
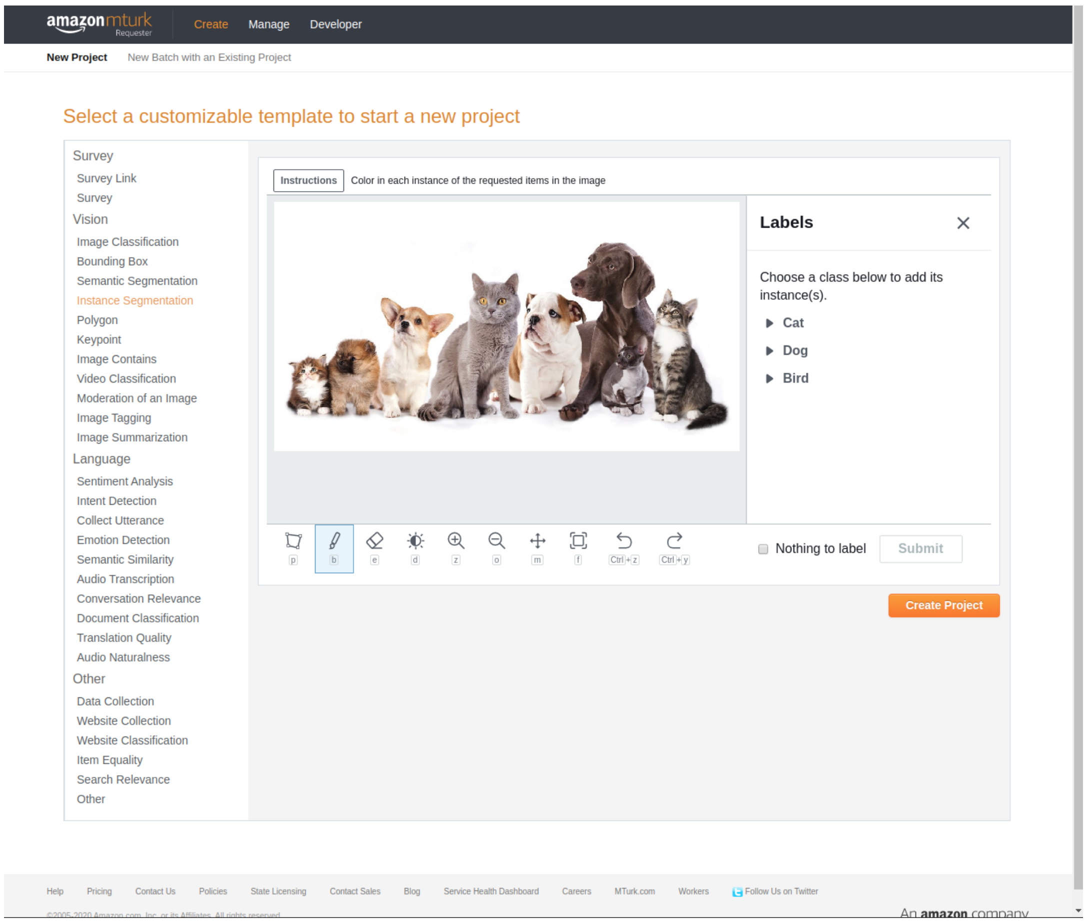


Figure 5.2: Images showing the workflow from a raw image towards a segmented image. (a) Shows a image taken by a participant. (b) Segments generated by a mean-shift algorithm. (c) Shows the original image overlayed by the borders of the different segments. (d) The final image after the user has clicked on the segments actually containing food. The light blue color indicates segments only containing food, while the darker blue color indicates a segments which contains food, but also background.

duration, when the job should expire, how many people should work on that job, the worker's qualifications, the price to pay the workers, and some details about the job. With all the information provided, the template can be customized in the next step by applying custom HTML. With that done, the job is ready to be published. To publish a job, one needs to tell MTurk where the images are hosted. MTurk again offers two possibilities for this. One could host all the images on an Amazon Cloud instance, or one can host the images on a custom server and provide a Comma-Separated Values (CSV) file, which contains the links to all the individual images and the classes to be expected in each image. As for this work, the images should not be hosted on an Amazon server, the second option was used, and the image URLs were provided.


Select a customizable template to start a new project

Color in each instance of the requested items in the image

Choose a class below to add its instance(s).

Figure 5.3: Overview over MTurk templates with an interactive preview, for instance, segmentation.

The templates from MTurk work well. However, the possibilities to customize the templates are very limited. For example, it is not possible to enable or disable any of the tools the template provides. When this part was implemented, the template always provided a bounding box tool, which enabled the users to draw a bounding box around the different instances instead of accurately creating the correct polygons. Nevertheless, the templates get continuously improved and provide a good

starting point for most of the projects. The second possibility is to host a custom web application, which can be some arbitrary HTML and JavaScript code. In this approach, MTurk only acts as a mediator. When workers accept a job, MTurk renders the provided custom interface into their framework and provides the custom code with a worker ID and a task ID. These IDs are then used to match the results of the custom interface to the corresponding user.
The possibility of creating a custom web application enables researchers to generate complex and very customized tasks. Nevertheless, creating those custom applications takes much time. To use the pre-segmentation, a custom application was created. This should make it easier and faster for the workers, and more importantly, it is easier to check if the workers have provided reasonable results.

To make the creation of custom applications less time consuming, there are various frameworks on Github, which take care of the communication between the web application and MTurk. This work was based on a framework found on Github[1]. This repository provides an example project and can be easily adjusted.
Another drawback of using custom web applications instead of the MTurk templates is that the MTurk evaluation tools cannot be used. The only possibility to check for the jobs is by using the MTurk API from a terminal. This means that one does not get a visualization of how many jobs have already been accepted and/or finished. This is a problem, as it is essential to keep track of the workers' results handed in. Moreover, jobs that were not completed need to be republished, which is impossible if one cannot see any information about the jobs. There exist various Github repositories, which create a visualization of all the tasks to solve this problem. In this project, the MTurk-Manager[2] repository has been used. This repository gathers information on all the published jobs. This includes the status of each job (accepted, pending, completed) and provides a way to download the results of the completed jobs. Furthermore, it offers the possibility to deny jobs if they do not match one's expectations.

5.1.4 *MTurk Sandbox*

To test the created job before publishing it to the workers, MTurk provides the ability to do so on a sandbox. This sandbox is visually and function-wise the same as the later production system, but the requester account has an unlimited amount of money, and if a worker accepts a job, he is not paid for it. Therefore, the only workers active on this platform are oneself, and the pilot users recruited to test the system. This adds the possibility to try out the task and see if everything works as expected. If the test is successful, then only one URL parameter needs to be changed to publish the production website's task. Although this works great for custom jobs, it does not work when using an MTurk template. When using an MTurk template, the job has to be recreated on the production website.

1 MTurk framework, `https://github.com/kimberli/mturk-template`, accessed: 26.07.2020
2 MTurk Manager, `https://github.com/jtjacques/mturk-manage`, accessed: 26.07.2020

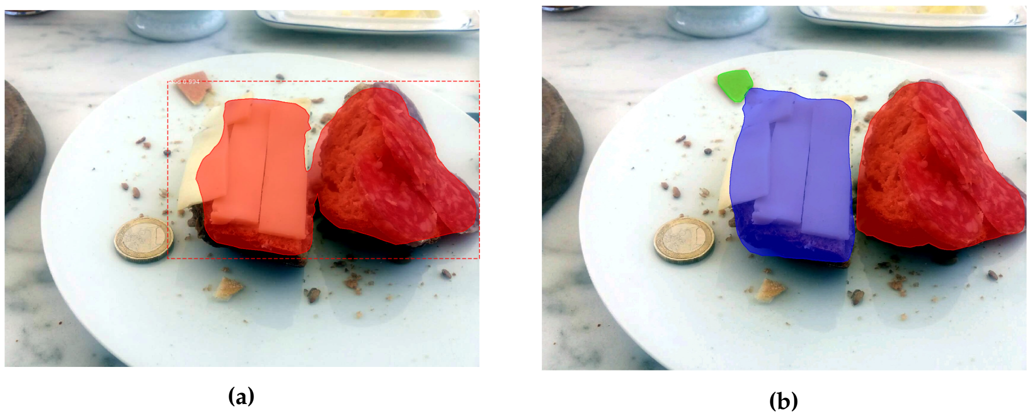


Figure 5.4: So far with the workflow presented a class segmentation is achieved (a). However the actual goal would be to have an instance segmentation (b)

5.1.5 *Class Segmentation vs Instance Segmentation*

The previously presented approach results in images similar looking to Figure 5.4 (a). These results represent a detection in two classes, a food- and a non-food class. Although this might already help when analyzing the images, more fine-granular detection is necessary. A first approach was to perform a connected component analysis on the gathered images (see Figure 5.4 (b)). Due to that, the unconnected components could be split into different instances. However, even with that approach, a plate with different food instances, which overlap, would still be detected as one instance of food, although it might contain multiple food occurrences. Therefore, a framework supporting the labeling of different food instances in one image is needed.
At the point this problem was encountered, MTurk already improved their labeling templates. For example, they removed the bounding box tool from the provided toolset. Furthermore, they added a template for polygon tasks. In this template, one can provide a predefined set of objects in the image, and the workers then segment the image, looking for those predefined objects. Due to the implementation of the BalancedEater App, the objects of the different images are already known. Only the location of the objects in the image is still missing. Thus, this new template fulfills all the required features, which are needed for this work. For that reason, the custom MTurk job has been discarded, and the predefined template from MTurk was used.

5.2 FROM MTURK TO ACTUAL TRAINING DATA

The outcome of the MTurk task is a CSV file with various different columns (see Figure 5.5). In these columns there are all the details about the job, for example, when the job was started, how long it took the worker to finish the job, the URL to the original image and the Base64 encoded segmented image. For the evaluation of the results two columns are of main interest. These are, the column about the

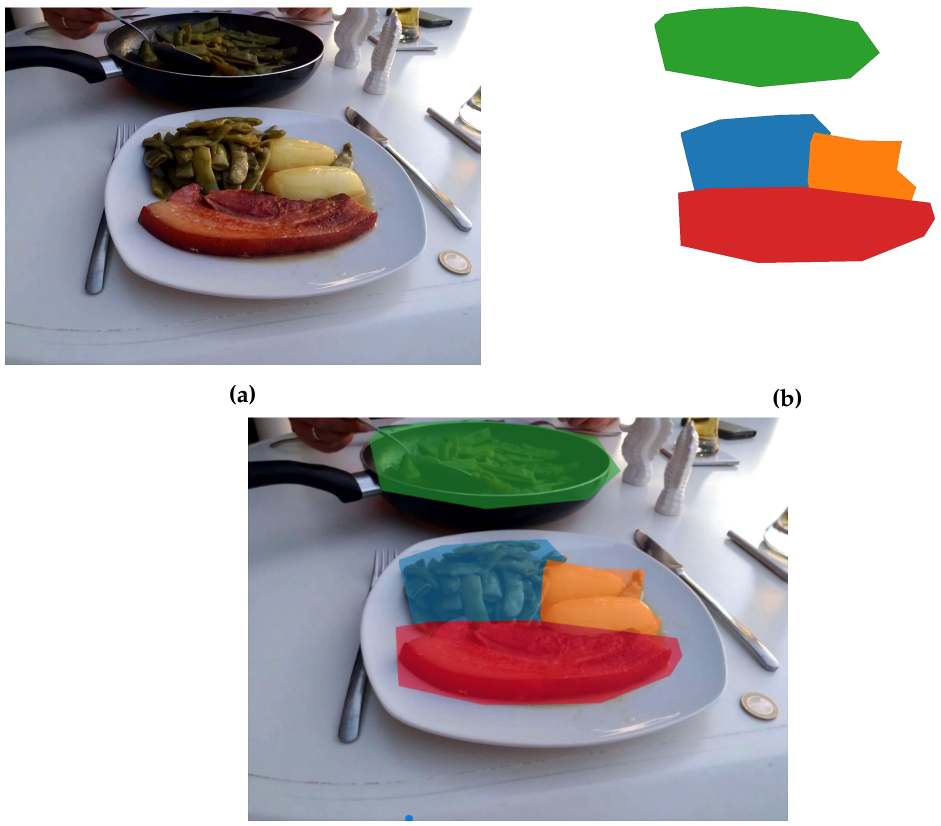

(a) (b) (c)

Figure 5.5: (a) Shows the original image, which is used as input to the MTurk task. (b) Shows a sample result of a worker. The Base64 string was transformed back into an image. (c) The resulting output after the MTurk task on top of the original image. The different colors indicate the single instances of food in the image

original image URL and the column, which includes the Base64 encoded result-image string.

5.2.1 *Mapping results to images*

As the actual resulting image is encoded into a string, it is hard to get an impression of the results. Therefore, a KNIME workflow was created, which maps this string back into an image. KNIME is a data analytics software, which can be used to analyze and transform data. This workflow uses the CSV-File created from the MTurk job and reads in all entries of the file. Afterward, for each entry, the Base64 string is transformed into an image, and for each different segment of food, a single image is created and saved to a predefined location. This is necessary, as the later used Mask-RCNN implementation needs a single image per food occurrence of the food image. Hence, the KNIME workflow also creates a folder for each original image containing all the single images of food occurrences. Additionally, the single occurrences are converted into binary images, where zero represents a non-food pixel, and a one represents a food pixel.

5.2.2 *Validation of the Results*

Validating the workers' results is an essential task, as the whole process can be interpreted as a MinMax problem from game theory. That means that it is related to a two-player game. While the requester wants to minimize the money and time spent on the job but wants to maximize the segments' quality, the worker wants to minimize his time spent but maximize the money gained. This means that there will always be some people who rush the jobs and try to do minimal work to receive the payment still. If the results are not checked, this could lead to empty segments for all images. Amazon tries to counter this by introducing a so-called "Hit Approval Rate". This rate defines for each worker how many requesters have denied their submitted jobs. When creating a job, a requester can add qualifications that the workers need to fulfill to accept the job. A reasonable value is to set the Hit Approval Rate to 95%. Hence, a requester can reject a submission if one submits an empty or not complete task. This results in the worker not getting paid and additionally loosing in his Hit Approval Rate. So whenever calculating the cost for the project, it should be kept in mind that there will be 10-20% of results, which cannot be used later.
The workers, on the other hand, cannot rate the requesters directly on MTurk. However, there are various blogs and forums where people will share insights about the posted jobs. If, for example, one's task takes a long time and one only pays poorly, they will report this back to the worker community, and many workers will not accept one's jobs in the future.
However, the question remains, on how one can detect the 10-20% of submissions which aren't performed correctly. Of course, empty submissions are easy to detect, but wrong or incomplete submissions are harder to find. A common approach is to have some self-labeled images in each job, which works as a validation set. If the submitted values of these validation images match the validation values, then the rest is more likely to be also correct.
This approach works if the worker has to select some specific segment in the image, but gets more challenging when the user can freely draw polygons. To make it still work, one needs to relax the constraint. Therefore, some metric needs to be introduced, which calculates the similarity of the two segments. In the project's case, this was the intersecting amount of pixels of the known segments and the segments submitted by the worker. After this intersection falls below a given threshold, a warning is put out, and the requester can have a closer look at that submission.
The problem with this approach is that their pixels need to be known to calculate the intersection for the segments. Nevertheless, after the worker's submission, only the polygon corners or the selected segments are known. As these polygon points or even the segments are just numbers, it is hard for the requester to check if the resulting numbers correspond to the images' correct pixels. Therefore, the KNIME workflow was expanded to transform the submitted values back onto the actual image and perform an automatic analysis of the validation images. See Figure 5.6 for the complete workflow.

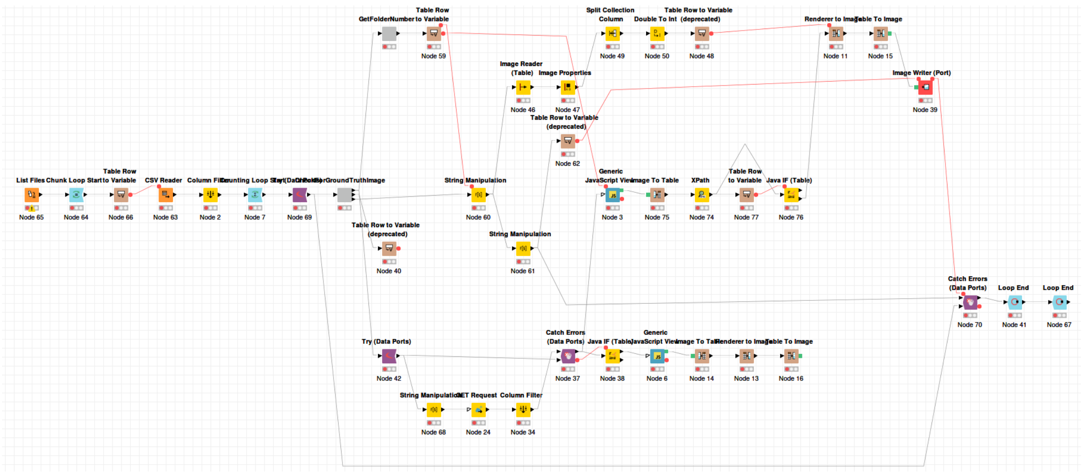


Figure 5.6: A KNIME workflow, which is used to load the different images of the MTurk job and transform the results back into images. Furthermore, an automatic assessment of the results is performed.

5.3 MASK R-CNN

In this section the used Neural Network and the training of this Neural Network are presented. The used network architecture is called Mask R-CNN. Mask R-CNN [26] is a further improvement of the Faster R-CNN algorithm. The main improvement of Mask R-CNN over Faster R-CNN is that in addition to having a bounding box and a class of the found occurrence as output, it adds a third output per-pixel basis mask of the found object. This is done based on the Faster R-CNN network by adding a third branch. This third branch consists of a fully connected network applied to each Region of Interest (RoI) to predict a segmentation mask. To get a precise enough alignment to make this per-pixel mask branch, mask R-CNN improves the RoI pooling layer from Fast-RCNN. In the paper, they call this improved version of the RoI pooling layer the RoIAlign layer. The RoIAlign layer intends to fix the location misalignment caused by quantization in the RoI pooling layer. An example output of Mask R-CNN is shown in Figure 5.7. It is impressive how precise the segmentation is, but it also shows some weaknesses, for example, in the bottom right, where the chair contains some pixels of the floor.

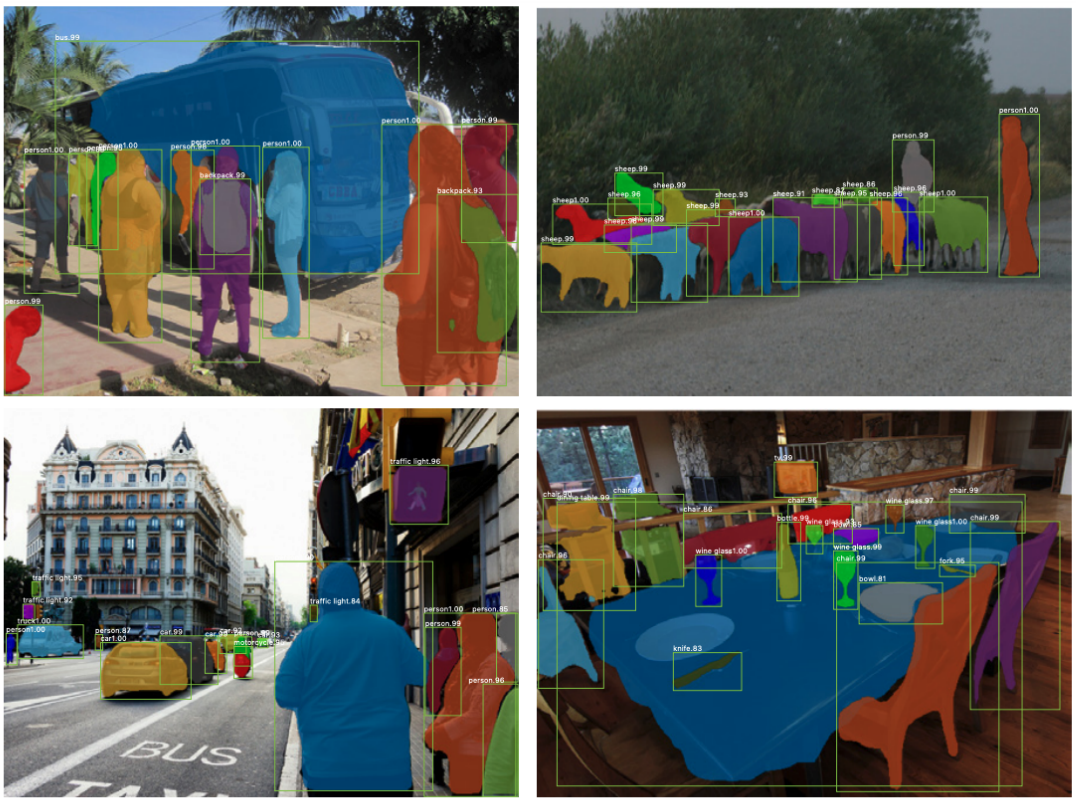


Figure 5.7: Example images of Mask R-CNN. Each found occurrence is marked by a bounding box, the mask pixels and a class with a confidence score.

5.3.1 *Implementation of Mask R-CNN*

There is an open-source Github project called Matterport, which provides a Python 3, Keras, and Tensorflow implementation of Mask R-CNN. The repository provides the following features:

F 1. *Source code of Mask R-CNN, which is built on Feature Pyramids and the ResNet101 architecture*

The complete source code of the project is open source and can be downloaded from the Github repository. The implementation is based on a feature Pyramid and ResNet101 implementation. This is good, as with ResNet, great levels of accuracy could be achieved.

F 2. *Training code for the MS COCO dataset*

The repository provides the code, which can retrain the model from the ground up from the MS COCO dataset. It can be useful to see how this works if something similar wants to be achived.

F 3. *Pre-trained weights for MS COCO*

A model with pre-trained weights for the MS COCO dataset is provided. This is important, as transfer learning is a common approach when dealing with small datasets. When performing transfer learning, a model previously trained on another task, where huge amounts of training data are available, is used, and only some parts of the network (the last layers) are retrained on the actual dataset. Therefore, it is good to have this already pre-trained model because training such a model on the complete dataset can take weeks.

F 4. *Jupyter notebooks to visualize the detection pipeline at every step*

For debugging and research purposes, it can be interesting to see the detection's visualizations in the different steps of the pipeline. The repository offers a Jupyter notebook, which enables exactly this.

F 5. *Parallel-Model class for multi-GPU training*

Training a model for a Neural Network is computationally costly. Therefore, it is important to have the algorithm run on a GPU and parallelize the process as much as possible. The repository has already been implemented.

F 6. *Evaluation of MS COCO metrics*

To evaluate a model's performance, it is important to have some evaluation metrics. The repository provides such a metric for the MS COCO dataset. This is not important when the network is trained on some custom dataset, which is different from the MS COCO dataset.

F 7. *Example of training the network on a custom dataset*

This is important when the plan is to use a custom dataset. As this is exactly what is needed for this work, it is useful, that the repository already provides an example on how to do that.

The framework makes it easy to get an implementation of Mask R-CNN running and already provides four different examples of how to use this framework. For this work, especially F1, F3, F4, F5, and F7 are important. F1 is necessary to expand the framework to one's custom needs and hence, is probably the most important feature. F3 is a great feature, as this allows to train a model on an already pre-trained model. This has the advantage that with only a few training images, good results can be achieved. F5 speeds up the network's training process when a computer with the availability of multiple GPUs is used. Finally, F7 is a good feature, as the framework already contains four different examples. These examples show how to train different networks on different datasets and give the developer an idea of how to implement it.

5.3.2 *Training the model*

After the changes are made, the framework is ready to train the Neural Network for the food segmentation task. As the training data is sparse and the repository provides a pre-trained model (see F3), this model is used as a foundation, and only the last layers are trained on the new data. As training is computationally heavy, the training process was run on a powerful computer, which includes four Nvidia Titan Xp[3] with 12 Gigabytes of memory each. The training was run for 30 epochs and took approximately two hours for the 353 training images. Although the amount of training data is scarce, the results already look promising (see Figure 5.8, for examples). In the examples, each of the uniformly colored parts of the image represent an instance found by the network. Additionally, a bounding box surrounds the instance, and on the top left, the found class (in this case, only food class is possible) and a score, a number representing the network's confidence for the part being of the food class, is displayed. In Figure 5.8 (a) for example, the individual parts of the meal are recognized well. Furthermore, the dark blue segment spans across the complete meal. Compared to that, in Figure 5.8 (b), the network cannot detect the fries as a single instance but can detect the burger. A possible explanation for this could be that a burger was present in the training images, but fries were not. This limitation could easily be improved by increasing the amount and variability of the training data.

3 Nvidia Titan Xp`https://www.nvidia.com/en-us/titan/titan-xp/`

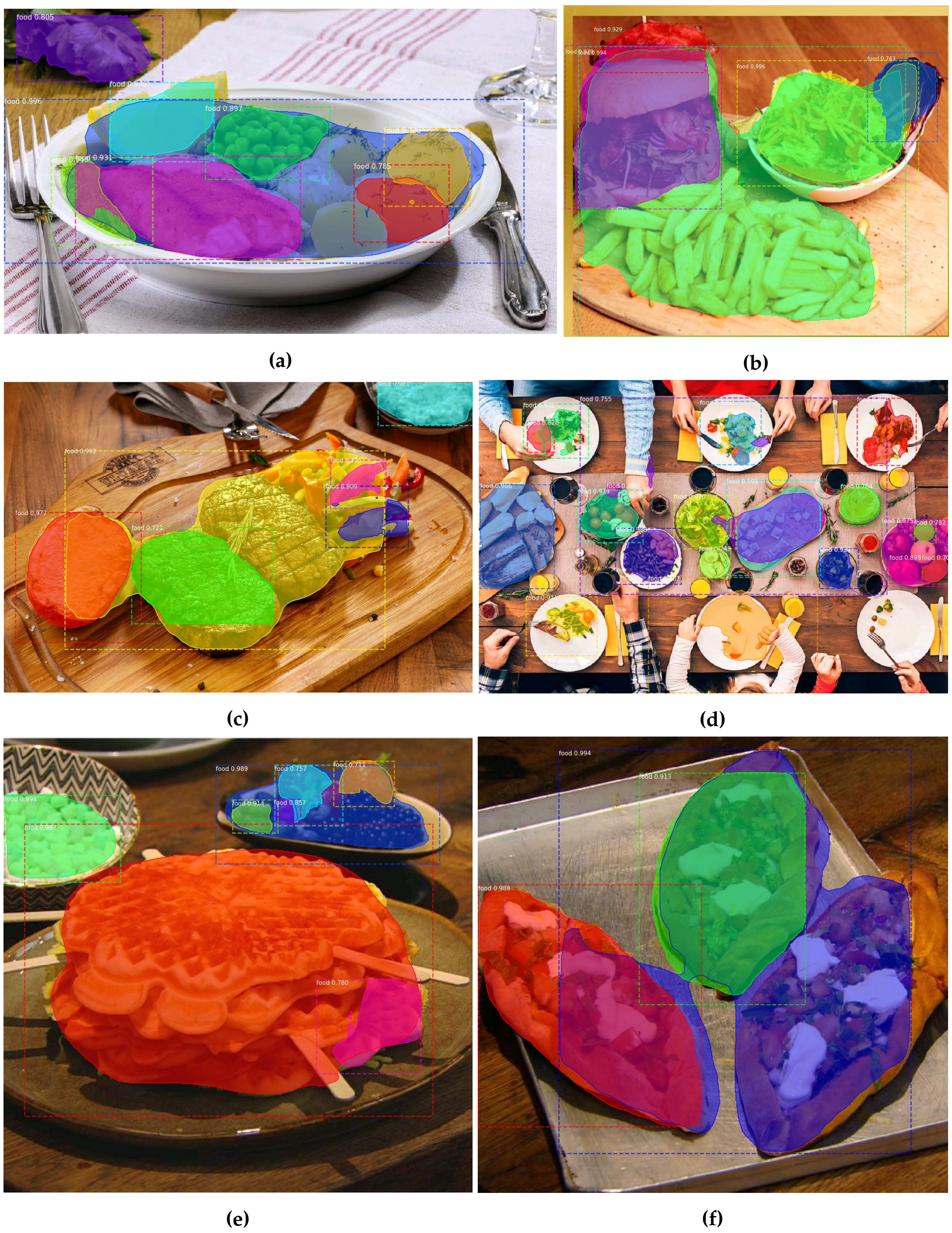


Figure 5.8: Examples of the segmentation result of the trained network. The individual segments are the colored parts of the image.

5.4 MOBILE APPLICATION

This section covers the last step of the pipeline. The two main parts are the uploading of the model to a server and communicating with a custom mobile application. This mobile application is used to upload taken food images to the server. On the server, the model is evaluated on the image, and the results are sent back to the mobile application (see Figure 5.9).

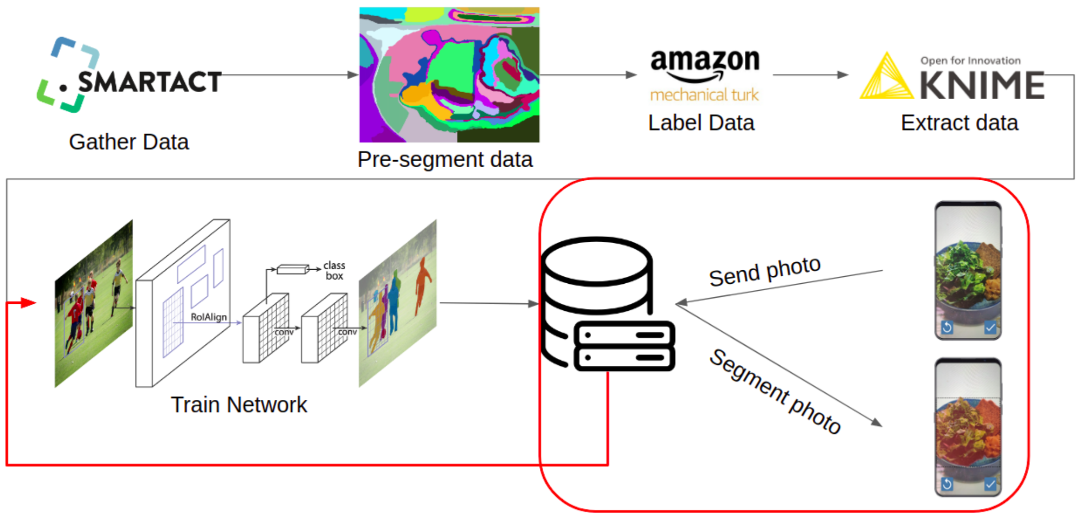


Figure 5.9: This section is about the structure, which is used to make the trained model available for users.

5.4.1 *Architecture*

Once the model is trained, it can predict an image's food parts. However, there is still the gap of how to enable users to use this trained model properly. Therefore, a system needs to be implemented, enabling the users to take pictures and get back the segmented results. There are two different approaches of how such a system could be implemented. The first one is to transform the trained model into a TensorFlow Lite model[4], which is an optimized format to run Neural Network models on a mobile device. This approach has the advantage that the model is evaluated directly on the mobile device, making it work offline and saving the time of sending the image to a server. However, this approach also has two major drawbacks. The first one is that if the trained model changes because it is improved on the newly gathered images, it needs to be updated on every mobile device. This update would require an online connection and updating service, which checks for the current version and the available version. The other drawback is that the TensorFlow Lite library does not yet support all TensorFlow operations, and there is no easy way to transform the model of the Mask R-CNN Matterport framework to a TensorFlow Lite model. Therefore, this approach could not be used in this work but could be further investigated in a future version.
The other possibility is to implement a server-client architecture. In this setup, the smartphone is used to take pictures of the meals. Then, the image is sent to a server. This server processes the incoming images and performs the segmentation on those images. Once this is done, the server sends back the resulting segments. Afterward, the smartphone receives and displays these results. This method's benefit is that because only the server evaluates the trained model on the incoming images, it is

4 TensorFlow Lite https://www.tensorflow.org/lite

easy to update that model. This has the advantage that the clients do not have to bother about any model updates. Furthermore, all the images are saved to a central location, making it easy to improve the model on the newly gathered images. The drawback of this method is that the sending of the images requires an online connection. Furthermore, the transfer of the image to the server and back takes time.

5.4.2 *Implementation*

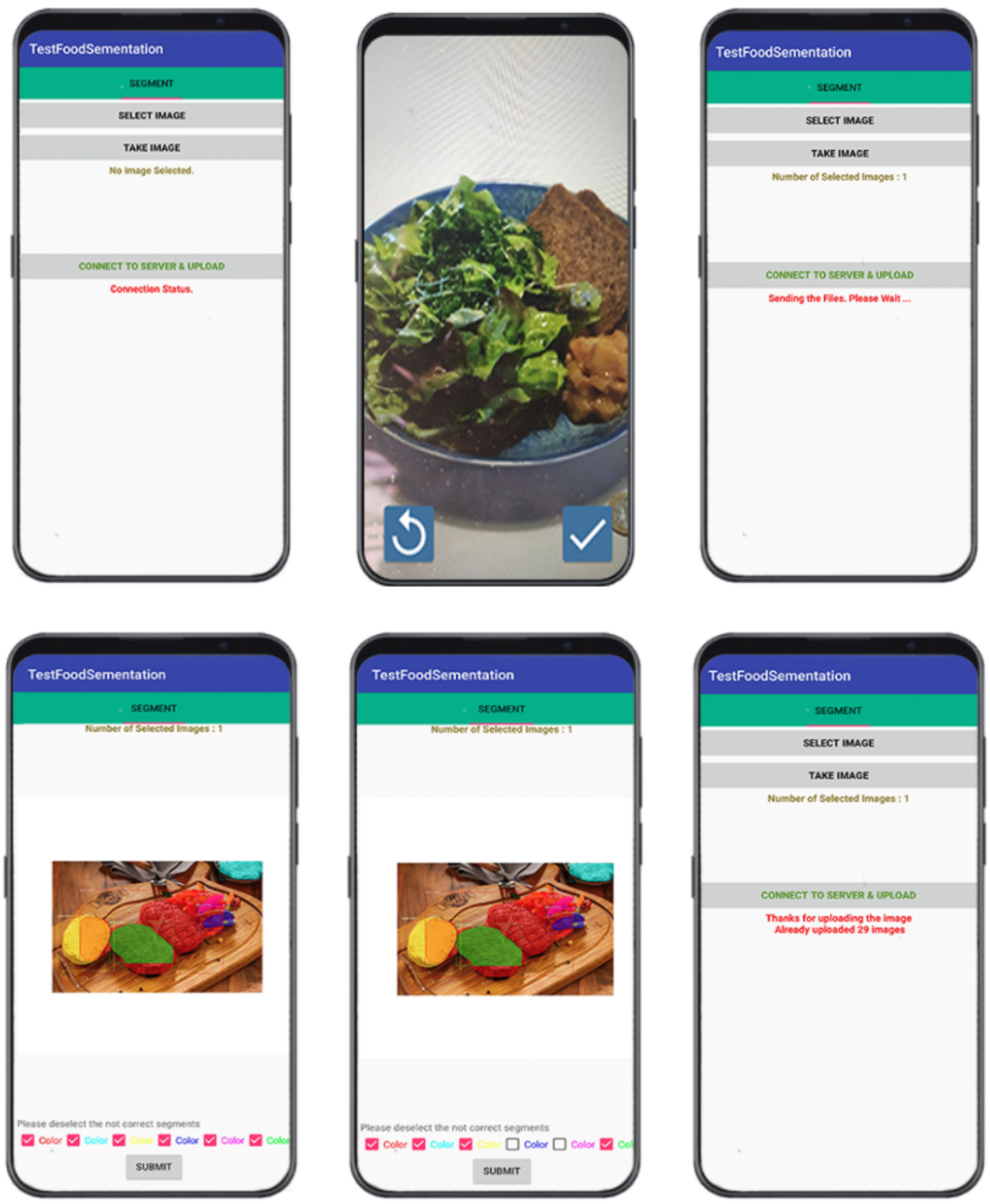
Because of the unavailability of a transformation from the Matterport model to a TensorFlow Lite model and the advantage of quickly updating the model, a server-client architecture was built. Furthermore to make it possible to improve the network on new images, the user can correct any errors made. This enhanced version of the image is then sent to the server and can be used to retrain the network.
The resulting application is depicted in Figure 5.10. When starting the application, the user has two possibilities. Either he provides an already taken image, or he takes a new one. After selecting/taking an image, the image is sent to the server. Depending on the image quality and the available internet connection, the process of sending and receiving the image takes approximately three to five seconds. When the results are received from the server, the original image is overlapped with the received segments. The user has the option to deselect the segments which are not correct, meaning not containing food. The resulting image is then sent back to the server, and the user gets displayed how many images he has already submitted to the server.

5.5 SUMMARY

This chapter described the creation of a pipeline, which can be used to get a Neural Network model as output given unlabeled training images as input. To achieve this, four steps are required, which consists of (1) getting the data labeled, (2) analyze and clean the results of the labeling, (3) train a Mask R-CNN model, and (4) finally deploy the trained model to a server, which serves a smartphone app (see Figure 8.1 for a graphic of the pipeline). To label the data, an Amazon MTurk service was implemented. In this service, workers are paid to segment the unlabeled data. The first idea of having a custom MTurk job with an individual user interface and pre-segmented images was discarded. The reason for this is that the pre-segmentation did not work for instance segmentation.
Furthermore, the MTurk templates improved during the project, and the custom job was no longer necessary, as the MTurk templates had all the required features. Although using the MTurk templates saves a lot of implementation time and provides superior user interfaces, it has a few drawbacks. Compared to a custom job, one cannot quickly assess the workers' quality in real-time while working on the job but needs to clean and assess the quality of the submitted results afterward. Therefore, a KNIME workflow was built, which automatically performs this quality assessment. To make this possible, some control images are injected into the train-

ing set, of which the ground truth values are already known. This measure is not perfect, and a manual check should still be performed from time to time. However, MTurk also has a rating system implemented, which should prevent the users from submitting only wrong results, as otherwise they could get blocked for other jobs. Once the data is cleaned, it is essential to save the images in a format, which the Mask R-CNN framework can use later on—for this work, saving each occurrence of food into a separate file and for each image creating a separate folder. This structure makes it easy for the framework to read in the ground truth values and train the network. With this done, the actual training of the network happens using the framework of the Matterport project. This framework provides some examples and hence makes it easy to adapt to customer needs. The outcome of this training is the Mask R-CNN network model, which is deployed to a server, which then can be used to communicate with a mobile application. With this mobile application, the user can take pictures and upload them to the server. The server will process the incoming images and transmits back the segmented results. Afterward, the users can remove the wrong segments and send back the resulting image to the server. With these new images on the server, the model could be retrained and improved over time.

Although this chapter already provides the functionality to achieve this pipeline, the pipeline's pieces could be further improved. The Matterport framework was adapted to match the custom needs for the food images, but the network parameters could be further tuned to get better results. Moreover, the mobile application to this point serves as a proof-of-concept and could be embedded into existing projects. With such an embedding, the app could be used in studies to speed up the process of tracking one's food intake. Such an embedding and the enhancement of the pipeline to perform not only segmentation but also classification is presented in the next chapter.



No Image Selected.

Connection Status.

Number of Selected Images : 1

Sending the Files. Please Wait ...

Please deselect the not correct segments

Thanks for uploading the image
Already uploaded 29 images

Figure 5.10: Workflow of the created mobile application, which is used to take and send food images to a server. The server processes these images and sends back the results to the app. The user can then deselect the wrong segments and sends back the final results to the server.

6 IMPLEMENTATION CLASSIFICATION

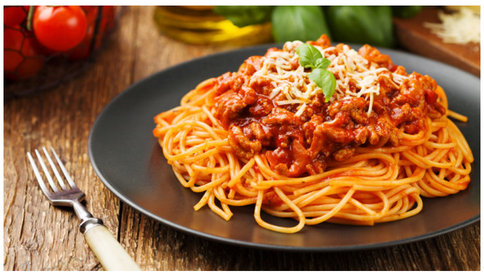
This chapter covers the improvements of the previously presented pipeline, which were implemented as part of this Thesis and are necessary to get a study prototype. One significant improvement was to enable the pipeline perform classification instead of only segmentation. Furthermore, the mobile app was integrated into the BalancedEater application. This application was built as part of the SMARTFOOD project and should help people to improve their eating behavior. The integration provides a more realistic scenario for the later evaluation of the system. In the first section the approach of the classification is described. Afterward, section 2 describes the steps needed to prepare the dataset to create the ground truth values. The Neural Network needs these ground truth values to learn the classes and position of the food parts in the images. After that, the used server architecture is described in section 3. This server architecture is then used in section 4 to train the model. Lastly, section 5 describes the integration of this model into a mobile application before the outcome of this chapter is summarized in section 6.

6.1 CLASSIFICATION

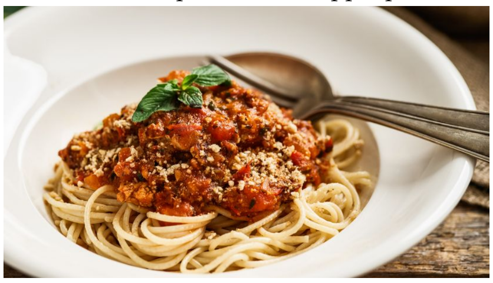
In the previous chapter, the segmentation pipeline was described. This pipeline enables to detect food parts on an image. However, for a everyday purpose, the food parts' position, and the actual names of the individual parts are interesting. For example, this could detect if a study participant is only eating sweets or if the food intake is balanced. This task seems simple for a human. However, it is only easy because humans are confronted with different kinds of meals every day, and therefore, learn all the different foods from an early age. The task of detecting food classes is challenging, as there are many different possible classes with high intraclass variability. This means that even images from the same class, showing the same food, can look very different.

(a) Picture of an apple pie [24]

(b) Another picture of an apple pie [1]



(c) Picture of a spaghetti bolognese meal [66]



(d) Picture of a spaghetti tofu meal [48]

Figure 6.1: Picture (a) and (b) show two different variants of apple pies. Picture (c) shows a picture of a spaghetti bolognese meal vs picture (d) shows a spaghetti tofu meal.

Figure 6.1 shows some example pictures which show this problem. Both images show an apple pie, but they look different. This shows the high intraclass variability of food images. Furthermore, there is another problem, while Figure 6.1(c) shows a spaghetti bolognese picture (d) shows a picture of spaghetti tofu. The food parts on both images look very similar, but the ingredients could be completely different when the food intake is analyzed. Furthermore, the problem gets more complex, as important parts of the meal could be occluded by other items. A common approach to counter this occlusion is to use generic classes to bundle similar things into the same group. This way, the pictures in Figure 6.1 (c) and (d) could be classified into the same group, which might be called spaghetti and sauce. Due to this generalization, some characteristics of the food parts are lost, but otherwise, the problem is unsolvable. In the Food101 dataset [5], for example, 101 classes are defined and for each of these classes an amount of 1000 pictures were gathered to counter the high intraclass variability. As the dataset used for this work consists of 20.000 pictures and the BalancedEater app applied the main classes of the Bundeslebensmittelschlüssel[1], the same classes were used for this implementation. These 26 classes are shown in Table 6.1.

6.2 GENERATING GROUND TRUTH VALUES

To train the Mask R-CNN network for these new classes the ground truth data needed to consist not only of the position of the meal, but additionally of the cor-

1 Bundeslebensmittelschlüssel https://blsdb.de/

Extras-Pastries	Extras-Salty	Extras-Sweets
Meat-Eggs	Meat-Meat Substitutes	Meat-Fish & Seafood
Meat-Sausage & Meat Products	Meat	Vegetable-Pulses
Vegetable-Mushrooms	Vegetables	Beverage-Alcoholic
Beverage-Non alcoholic	Cereals-Bread	Cereals-Grain
Cereals-Potatoes	Cereals-Pasta	Milk-Dairy products
Milk-Cheese	Milk	Fruit-Fruits
Fruit-Nuts	Other-Sauce	Other-Soup
Other-Composed		

Table 6.1: This tables consists of all the used classes

rect class of the individual parts. Therefore, the Amazon MTurk Job described in the previous chapter needs to be enhanced to make it possible for the worker to add a class to each food segment in the image. Conveniently for every image in the gathered food dataset the displayed food classes are known. The workers could only choose between these predefined classes for each picture. This should prevent the workers from choosing wrong classes for the food parts. Linking the correct classes to the images was realized with the help of a second KNIME workflow, which is shown in Figure 6.2. As a first step, the workflow gathers all the available datasets for the images collected during the different studies in the SMARTFOOD project. Afterwards, it connects the available food classes to the individual images and translates the classes from German to English. The outcome of this workflow is a CSV file, that consists of the mapping of the food classes to the individual images.

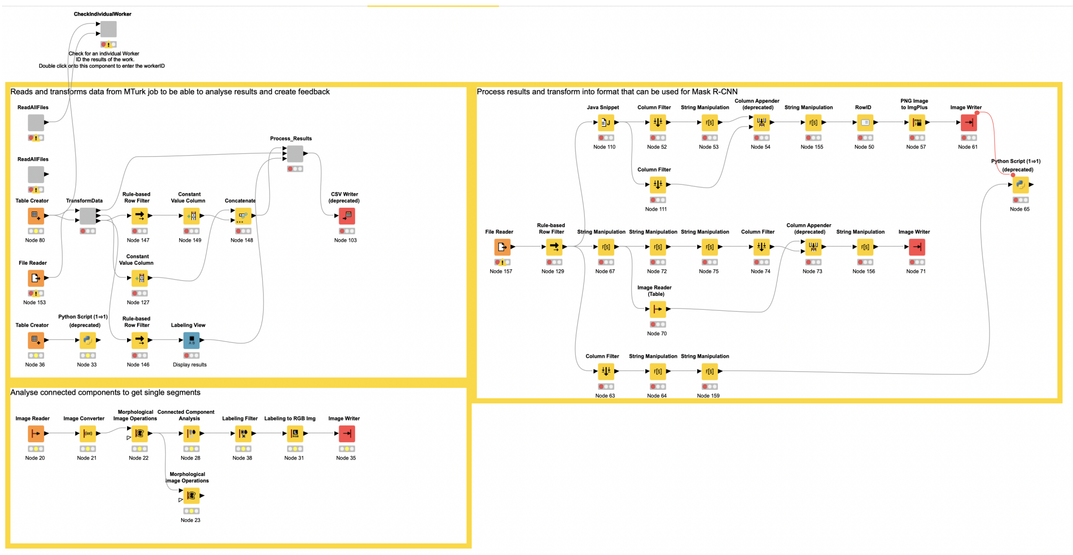
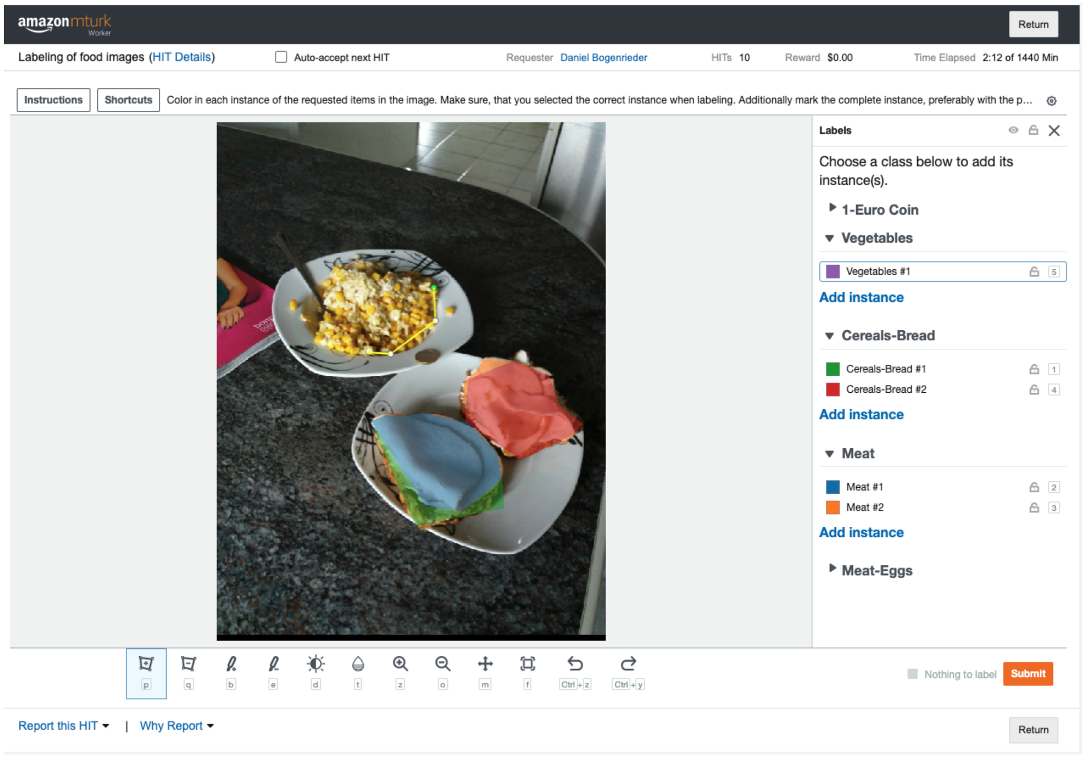


Figure 6.2: Workflow to crawl all images of the previously run studies and connect each of the images with the available food classes.

The previously created CSV file is then used as input data for the MTurk-Job. As already used in the previous chapter, a predefined instance segmentation job of MTurk was deployed. This enables the workers to see the available classes and make use of the labeling tools of the MTurk framework. It allows the workers to create a new instance of food class for every part of the image. With either a polygon-tool or a painting tool, the workers can mark the corresponding parts in the image (see Figure 6.3). After completing the labeling of one image, the workers can label another image or stop labeling images. The dataset for this work consists of 17.804 unlabeled images, and to ensure quality of the results, every image was labeled twice. Therefore, an amount of 35.608 images needed labeling. It took about seven days for the workers to complete the tasks.


Labeling of food images (HIT Details) — Auto-accept next HIT — Requester Daniel Bogenrieder — HITs 10 — Reward $0.00 — Time Elapsed 2:12 of 1440 Min

Color in each instance of the requested items in the image. Make sure, that you selected the correct instance when labeling. Additionally mark the complete instance, preferably with the p...

Labels

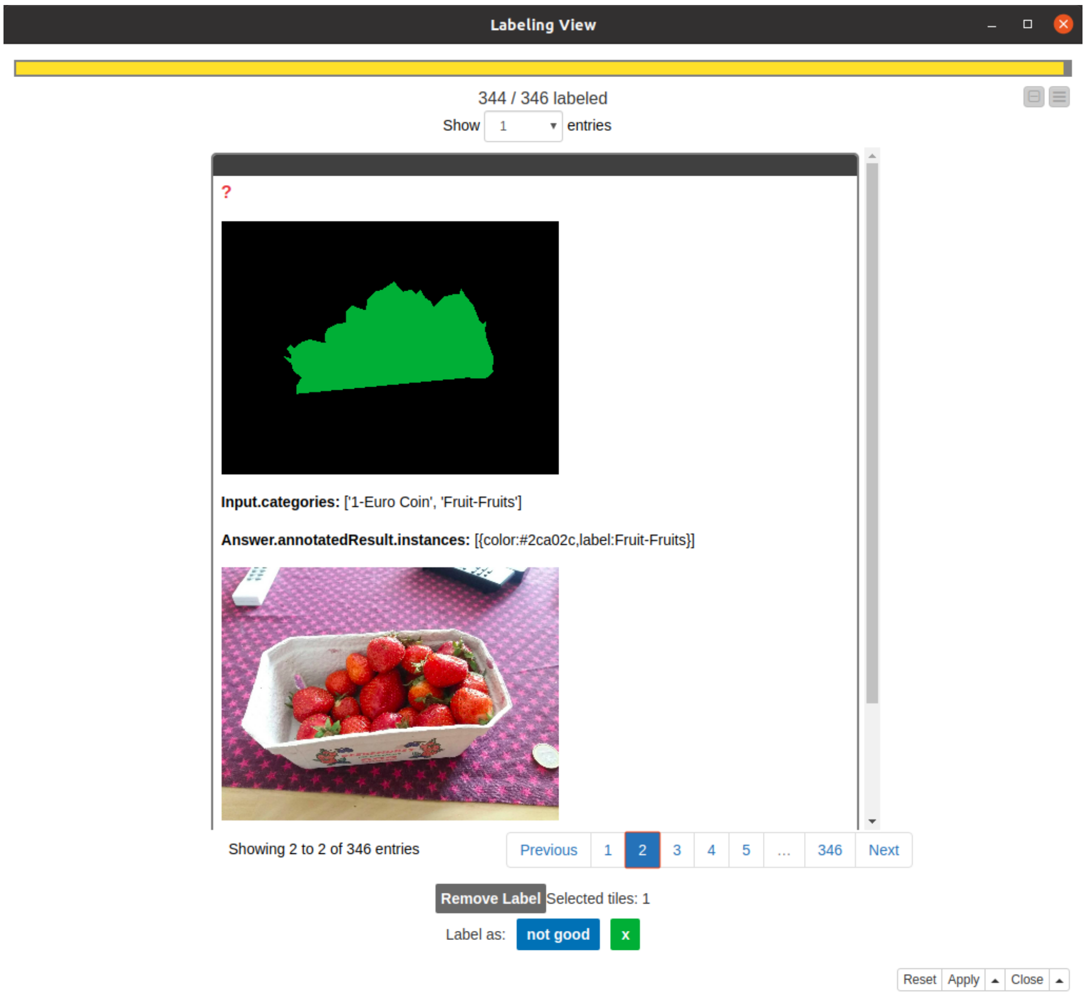
Choose a class below to add its instance(s).

- 1-Euro Coin
- Vegetables
  - Vegetables #1
- Cereals-Bread
  - Cereals-Bread #1
  - Cereals-Bread #2
- Meat
  - Meat #1
  - Meat #2
- Meat-Eggs

Figure 6.3: This figure shows an example of the implemented MTurk job. The task of the worker is to select and label the individual food instances of the provided image. On the right, the different instances of the classes can be seen, and on the left, in the image, the color regions show the marked food parts.

The MTurk job results are a huge CSV-file, which only contains the polygon data of the labels. To be able to manually check if the data is correct, the results need to be converted into images. Another KNIME workflow was used to perform this transformation. Therefore, the workflow downloads the original image from the database and overlays it with the labeling results. In the first step, the images with less than two created classes are filtered out. This number is based on the observation that if only one or zero classes were labeled, workers often labeled poorly. Additionally, 5% of the other images were filtered out to additionally check this subset of remaining images. This subset is put into a KNIME labeling node (see Figure 6.4). The node

visualizes the results of the labeling job and allows to mark good or bad results. In this work, this task was done manually by the author, but it could be outsourced in another MTurk task and let other workers decide if the quality of the previous labeling job is good or not.
Once the manual review is done, the result is a CSV-file with a review column appended. The resulting file is uploaded to MTurk, where the jobs get rejected or approved based on the decision made in the labeling node. MTurk will recreate the rejected jobs, and the workers will hopefully create better results. This process is repeated until satisfactory results are achieved.
To process the data with the Mask R-CNN framework, each image needs to be split into a single image for every food instance. This split is performed with a Python script inside KNIME. Additionally, a CSV-file is created, which saves the mapping of the created images and the corresponding food class. In this CSV-file one image can be included multiple times showing each time a different food segment


Labeling View
344 / 346 labeled
Show 1 entries
?
Showing 2 to 2 of 346 entries
Previous 1 2 3 4 5 … 346 Next
Remove Label Selected tiles: 1
Label as: not good x
Reset Apply Close

Input.categories: ['1-Euro Coin', 'Fruit-Fruits']

Answer.annotatedResult.instances: [{color:#2ca02c,label:Fruit-Fruits}]

Figure 6.4: This figure shows the used KNIME Labeling View. In this view, all the available classes and the results of the MTurk task are shown. The two buttons at the bottom can than be used to decide whether the result is good or not.

6.2.1 *Lessons Learned when working with MTurk*

While working with MTurk, some general guidelines could be identified:

- The task's instructions are the most important piece of the task and should be carefully designed. Although the instructions have to be very specific about the wanted outcome, they also have to be as short as possible, as otherwise, the workers will not read them. A common approach is to use small videos or images to explain the task and the wanted outcome. Additionally, it is also important to include some images/descriptions of good and bad examples of the outcome.
- The quality varies a lot and some quality auditing needs to be implemented. There are always some workers who try to bypass the task and receive the money. This needs to either be prevented in the task design itself, or reject those hits afterward during the manual assessment. Even if this prevention is implemented, it is advisable to calculate with some "lost" images.
- Workers care about their MTurk reputation. When a job is rejected, their rating drops, and they might not be able to take other jobs. Furthermore, giving an explanation (of why their job was rejected) is crucial to avoid multiple messages to the creator of the job. As workers have their places where they rate the jobs, it might be beneficial to accept jobs, which are almost correct, or even jobs where it is clear that they tried it. This will make them write positively about the job, and additional workers will work on the job.
- The analysis of the results can get time-consuming, as the results need to be processed, and the amount of data is large. Additionally, manually checking the quality of the images takes a lot of time.
- In general, the speed of the workers completing the jobs is fast. However, this, of course, heavily depends on the provided task and the payment.

6.3 SERVER ARCHITECTURE

Training Neural Networks takes a considerable amount of time, depending on the available computational power. As the gathered dataset is large, the computation used a computational cluster. This cluster consists of multiple GPUs and powerful CPUs in addition to a large memory. To train the network in the cluster, the setup had to be wrapped into a Docker container [2]. In this work, the container includes all the necessary packages and the actual code to train the network. The advantage of using Docker is that the resulting container can be downloaded and started from any computer, which has Docker installed and, therefore, is not depending on the operation system or any installed packages.
On the cluster, there is an additional layer, which is there to manage all the resources. This layer or service is called Kubernetes [3]. Kubernetes is an open-source

2 Docker `https://www.Docker.com/`
3 Kubernetes `https://kubernetes.io/`

platform initially developed by Google. It is used to automate, manage, and scale containerized workloads and services. To run the Docker container on the Kubernetes cluster, a Kubernetes job has to be created. This job consists of a configuration file that sets the settings of the Kubernetes job. This includes the maximum amount of memory, the number of graphic cards used, the storage requirements of the job, and the path to the Docker image. The configuration file used for this work's training job is shown in Source Code 6.1. It shows that a minimum of 16 gigabytes and a maximum of 128 gigabytes of memory are used for this job. Additionally, only one GPU, and persistent storage is used. This is important, as otherwise the produced models of the training would be deleted after the Kubernetes job finishes.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  # name of the job
  # you should prepend your username so the the ID becomes unique.
  # note that you might share a namespace with many other users.
  name: 'Name of the Job'
spec:
  template:
    spec:
      # List of containers belonging to the job starts here
      containers:
      # container name used for pod creation
      - name: 'Container Name'
        # container image from the registry
        image: 'Docker image adress'
        resources:
          requests:
            # this gives us a minimum 16 GiB of main memory to work with.
            memory: "16Gi"
          # limits are maximum resource allocations
          limits:
            # this gives an absolute limit of 128 GiB of main memory.
            # exceeding it will mean the container exits immediately with an error.
            memory: "128Gi"
            # this requests a number of GPUs. GPUs will be allocated to the container
            # exclusively. No fractional GPUs can be requested.
            nvidia.com/gpu: "1"
        # Environment variables set when running the image,
        # which can for example used to configure your application.
        env:
        - name: PYTHONUNBUFFERED
          value: "1"
        # the command which is executed after container creation
        command: ["/application/run.sh"]
        # list of mount paths within the container which will be
```

```
   # bound to persistent volumes.
   volumeMounts:
   - mountPath: "/application/mask_rcnn/logs"
     # name of the volume for this path (from the below list)
     name: pvc-mask-rcnn-logs
  # login credentials to the docker registry.
  # for convenience, a readonly credential is provided as a secret in each namespa
  imagePullSecrets:
  - name: registry-ro-login
  # containers will never restart
  restartPolicy: Never
  volumes:
   # User-defined name of the persistent volume within this configuration.
   - name: pvc-mask-rcnn-logs
     persistentVolumeClaim:
      # name of the PVC this volume binds to
      claimName: 'Name of the volume'
# number of retries after failure.
# since we typically have to fix something in this case, set to zero by default.
backoffLimit: 0
```

Source Code 6.1: Configuration of the training Kubernetes job

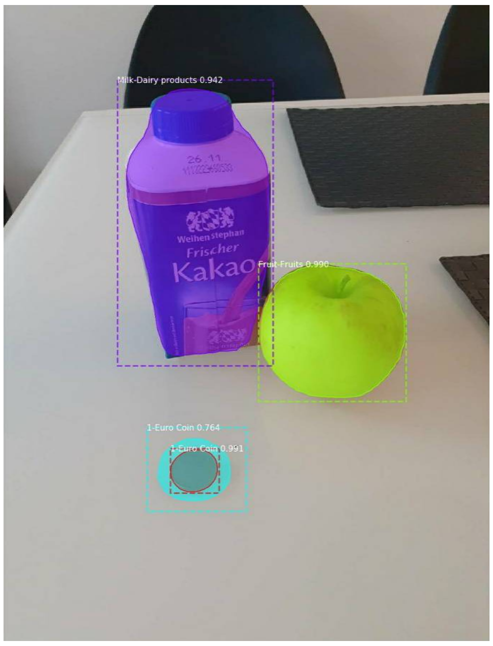
6.4 TRAINING THE MODELS

The Matterport framework handles the actual training of the network. The pre-trained model of the Matterport repository is used as a basis, and only the last layer of the network is retrained with the new images. The training was run for 472 epochs and took approximately one week for the 20.000 training images. The network performs data augmentation to also detect screwed and rotated images. Data augmentation is a technique, where some pre-defined percentage of the input images are randomly screwed and rotated. This way, the network will also learn the features for those images. The achieved results look promising (see Figure 6.5, for examples).

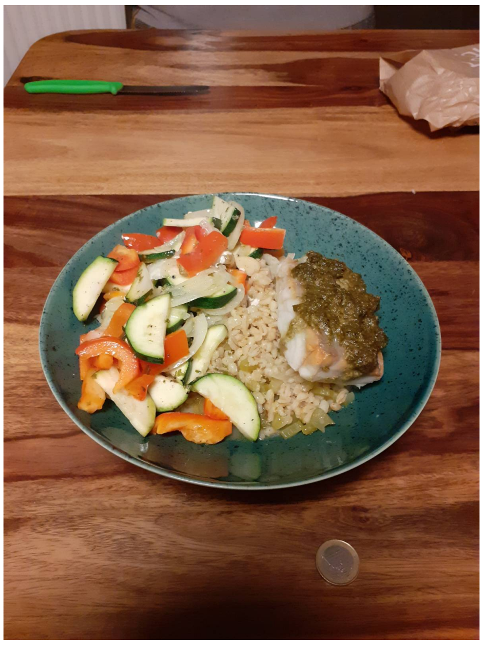
6.5 MOBILE APPLICATION

To create a useful application, which can be used in a realistic scenario, the created framework was embedded into the BalancedEater app. The BalancedEater app is an app created in the SMARTFOOD project and supports users in assessing their food intake (see Figure 6.6). The participants need to take a photo of their meals and label what food types are on that photo. This work aims to integrate the trained network into this particular step of the food labeling and automate this process. Therefore, the app will send the image to the Kubernetes service, which then analyzes the image and returns the found classes and segments. These results are then displayed to the user, and the user can edit these results if they are not correct (see Figure 6.6

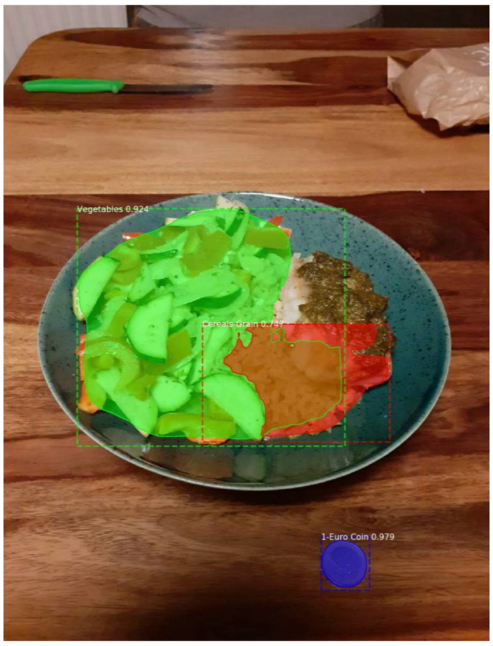
(a)



(b)



(c)



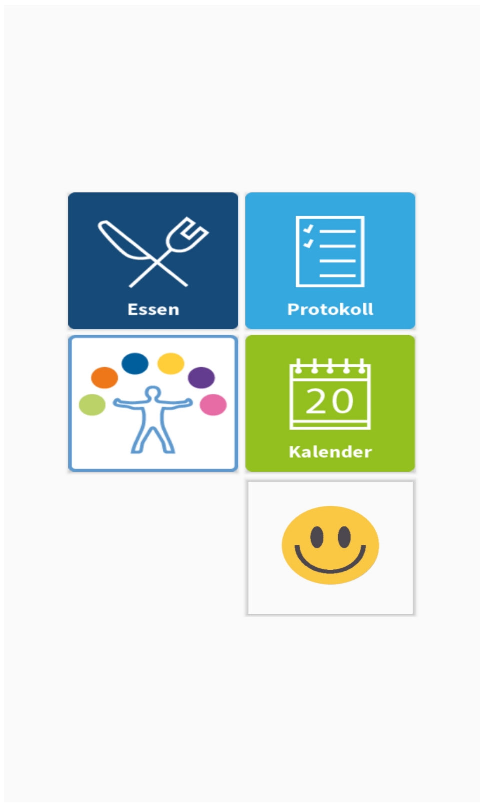
(d)

Figure 6.5: Picture (a) shows the raw image of a milk package and an apple (b) shows the results of the analysed image. Both objects are detected perfectly. Picture (c) shows a plate with a mixed dish. Although the fish is not detected at all, the vegetables and the rice are detected nicely.

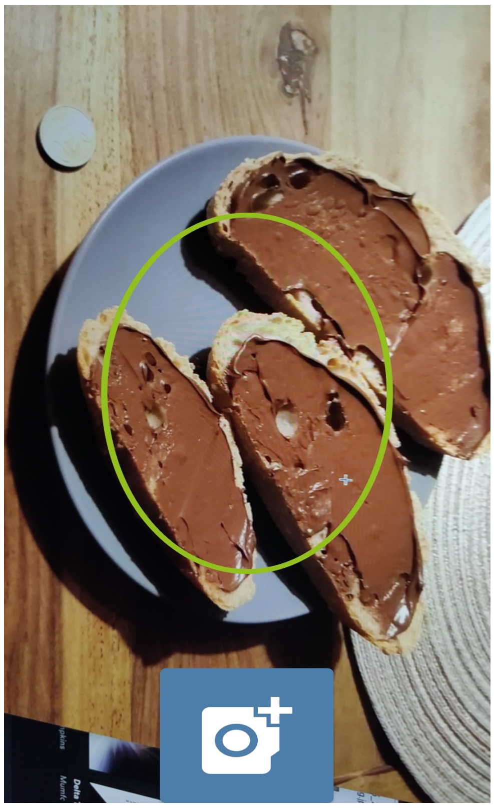
c)). Furthermore, the participants can click on the resulting image on the top to get a larger version of this image (see Figure 6.6 d)). In the image, the detected food parts are surrounded by a dotted box, and on the top left corner of this box, the suggested food class is shown.
The app also provides feedback to the users on how balanced the food intake is. This feature was kept in the app to create a more realistic experience but is not crucial for evaluating the tested framework.

6.6 SUMMARY

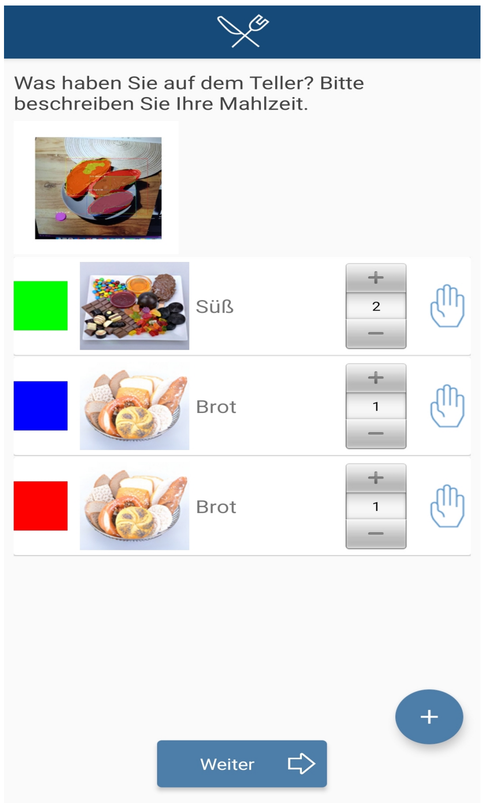
In this chapter, the added improvements to perform a classification were presented. One crucial step was to decide on the classes that want to be detected. As the BalancedEater app is based on the Bundeslebensmittel Schüssel; the same 25 classes were used. These classes are high-level food classes and do not determine the exact food type, but can already provide details about the participants' balance of food intake.
Furthermore, the MTurk task was adjusted to make use of the known food items in the images and let the workers mark these food items' positions. To analyze the results of this job, the KNIME workflow was improved to use some simple heuristics to find the wrong labeled images.
The training of the network was moved from one computer into a computation cluster using Docker and Kubernetes. That way, the training can easily be scaled to use more computational resources or train multiple models in parallel. To later evaluate the performance of the model in a realistic study set up, the system was integrated into the BalancedEater app. This integration makes a comparison of the existing BalancedEater app with the improved version, including an automatic analysis of the food images, possible.



(a)



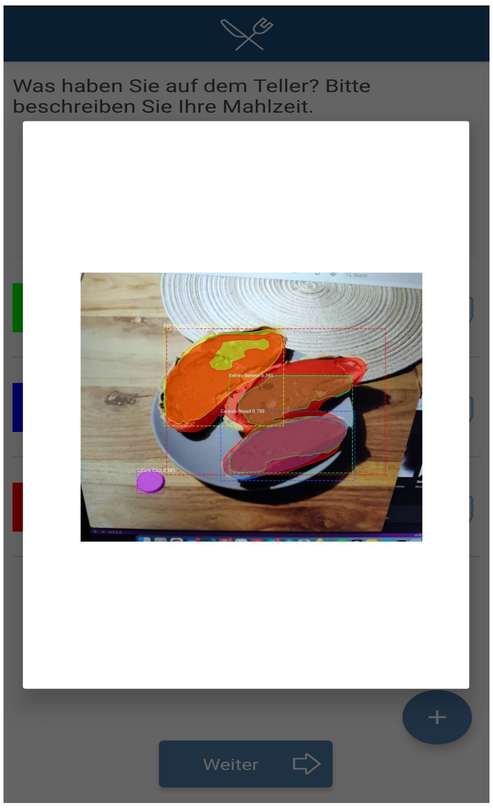
(b)



Was haben Sie auf dem Teller? Bitte beschreiben Sie Ihre Mahlzeit.

🟩	Süß	2
🟦	Brot	1
🟥	Brot	1

(c)



Was haben Sie auf dem Teller? Bitte beschreiben Sie Ihre Mahlzeit.

(d)

Figure 6.6: Screenshots of the modified BalancedEater app. Picture (a) shows the start screen of the app. After selecting a meal type a photo of the meal has to be taken (b). (c) shows the analysis of the meal. The network detected two different instances of bread and one instance of sweets. (d) shows the zoomed in view of the analysis. This enables the user to better see the segmentation of the image. Each occurrence of food is surrounded by a dotted box and on the top left corner the detected class is shown.

7 SYSTEM EVALUATION

The developed network integrated in the BalancedEater application allows users to automatically detect the food classes on their meal. This should speed up the documentation of a participants food intake, which was one main goal of this thesis. A study is conducted to evaluate the usability and accuracy of the model. Thereby the research question presented in section 1 are of main focus. Afterward, in the second chapter the design of the study and the used mobile application are presented. In section 3 the study results are elaborated before they are discussed in section 4.

7.1 RESEARCH QUESTIONS & HYPOTHESIS

The following research questions have mainly arisen from this master thesis's goal, which is to speed up the tracking of the food intake while improving the user experience of the app or at least keeping the experience constant. In the following the four research questions are listed and explained.

RQ 1. *Efficiency: Can automatic food classification speed up tracking time compared to manually tracking the food intake?*

To examine if this has been achieved, the created app tracks the time it takes users to input their food intake.

RQ 2. *Task Load: Is there a difference in the task load when using the automated system compared to the manual system?*

The task load can give a hint on how exhausting it is to use the app. It is crucial to keep it low, as otherwise, users might not use the app even though it would speed up the food intake tracking. To keep track of the task load after each meal, the participants have filled out a NASA-TLX questionnaire. Furthermore, after the study, questions regarding the subjective perception were asked in a questionnaire.

RQ 3. *Usability: Do users perceive a difference between the two input systems?*

This question aims to find out if the usability of the system is affected by the automatic analysis. If, for example, the processing time of the automatic approach is taking too long, users might be annoyed, which would influence the user experience. Therefore, to answer this question, the processing time and the final questionnaire are the leading indicators.

RQ 4. *Quality: Is the quality of the segmentation and the classification high enough to be used in a real-life application?*

The automatic analysis of the food images will only work correctly once the segmentation and classification quality is good enough. It is hard to tell when this "good enough" is reached. Therefore, the answer to this question involves two measures. At first, the participants are asked about the segmentation and classification quality after each meal tracking to get a subjective measure. Second, a performance test of the model was run to test the model on unseen images to get an objective measure. Therefore, a subset of the labeled dataset was not used for the training of the network. The trained model was then used to predict the classes and the segments on these images and the accuracy of the network could be calculated.
This work is based on the following two hypothesis:

H 1. *That the automatic condition will speed up the tracking time of the participants.*

The participants will need less time to input the pieces of the meal, therefore it is expected, that the automatic condition will reduce the tracking time in comparison to the manual condition.

H 2. *The task load of the users is reduced and the user experience is increased, when using the automated condition.*

With the automatic detection the participants do not need to enter the individual pieces of their meals, therefore it is expected, that the task load is reduced and the usability is increased. To investigate these hypothesis, the dependent variables of the tracking time and the task load as well as the usability of the system are tracked.

7.2 FIELD STUDY

In the following, the study design will be presented. As the study was a field study, the participants tested the system in their everyday life. The study was run during the COVID-19 pandemic, so the participants took the images mainly at their homes. The participants might have different eating behaviors in home office than at their actual workplace, but this should not affect the study much. For this work, this means, that more self-cooked images were taken. Although the study app runs on every Android phone above Android 4.4, all participants received a study smartphone, which makes sure that the app's look and feel is the same for everyone. The chosen smartphone was a Samsung Galaxy A20e[1].

7.2.1 *Participants*

16 participants took part in the study, but the app was not working for one participant due to connection issues. Therefore, the data of 15 participants could be us for the evaluation. The age of the participants ranged from 19 to 55 years with an average of 28 years (M = 25, SD = 8.4). 12 of the 15 participants were female, and three were male. Regarding their experience with the BalancedEater applica-

1 Samsung Galaxy A20e https://www.samsung.com/de/smartphones/galaxy-a/galaxy-a20e-black-32gb-sm-a202fzkddbt/

tion, three of the participant answered that they have never seen it before. Seven have seen the app already but barely used it. Two have seen the app already and used it a little, and three have extensively used the app. This shows that most of the participants where already familiar with the BalancedEater application. All of the participants are either from the psychology or computer science department, which were involved in developing the original app. The reason for choosing only expert users was, to investigate if the quality of the classification and segmentation is high enough for the experts.

7.2.2 *Procedure*

The study consisted of two different conditions (a manual tracking and an automatic classification) and each of these conditions lasted four days (see Figure 7.1). The participants either started with the manual or the automatic classification. While the participants needed to input the food classes themselves in the manual classification, an artificial intelligence tried to determine what is in the food images in the automatic classification. In the automatic classification, the participants could still delete wrongly detected parts and add missing ones.
To give every participant the same information, there was an online briefing meeting. In this meeting, the goals and the structure of the study was explained. Additionally, the participants could ask questions, and the demographic questionnaire was filled out. The primary purpose of this briefing was to explain the app to the participants. The main focus was the explanation of the different categories in which the foods should be classified. These categories are the 25 categories already described in Table 6.1. As classifying the food into these main categories might be difficult, it was essential to give the participants examples of which meals correspond to what category (see Table 7.1). For all participants, the study started simultaneously and, therefore, also ended at the same time. The conditions were counterbalanced, which means that half of the participants started with the automatic classification condition, while the other half started with the manual classification condition. After four days, the conditions switched and the participants were notified.
Additionally, on the fourth day the System Usability Scale (SUS)-questionnaire showed up on their main screen. After filling it out, the study continued with the other condition for the other four days. At the end of the study, the participants received another SUS-questionnaire and three questions asking about the subjective estimation of the different conditions. As the study was started on a Wednesday and lasted for eight days, every condition included one day of the weekend. This is important, as the participants' eating behavior might be different on the weekend compared to a typical working day.
The participants' task was to take photos of every meal they ate and either correct the automatic classification or manually enter the different food parts. Furthermore, they needed to fill out a NASA-TLX questionnaire after every meal-tracking. The workflow of a food tracking task is shown in Figure 6.6

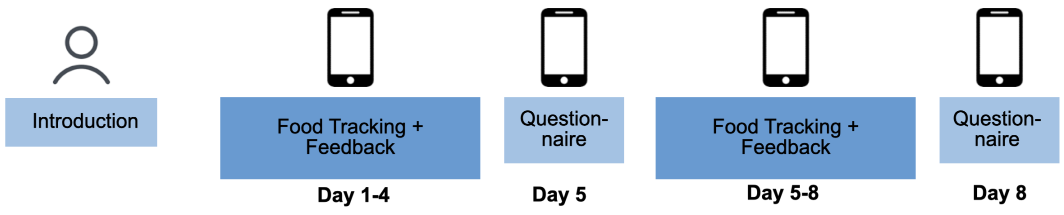


Figure 7.1: Timeline of the study. After the briefing, there are four days with either the automatic or the manual tracking. After the four days the condition switches for another four days.

Meal	Food Components
Bread with Cheese	Bread + Milk Products (Butter) + Cheese
Couscous Salad	Wheat (Couscous) + Vegetables + Cheese
Schnitzel with Fries	Meat (Schnitzel) + Potatoes (fries)
Almond-/ Soy **Milk**	Milk
Soy **Yogurt**	Milk products
Meat **substitutes**	Meat substitutes

Table 7.1: This table gives some examples of meals and the corresponding meal parts. The food components are based on high level food categories

7.2.3 *Data Gathering*

To gather all the relevant information during the study six different approaches have been implemented. A demographic questionnaire, a NASA-TLX questionnaire, a SUS-questionnaire, the time taken for the classification, a final questionnaire and in general all change events inside the app. The demographic questionnaire is displayed at the first start of the app and is used to gather the demographic data of the participants. With the NASA-TLX the task load of the participants is tracked. This questionnaire is shown after each meal tracking. The usability of the system is evaluated with the SUS-Questionnaire. It is shown at after the first condition is over (day four) and after the study has ended (day eight). To evaluate if the automatic tracking could improve the time taken to track the meals, the tracking time is logged. At the end of the study, additional to the SUS-Questionnaire four open questions were asked to get subjective feedback on the usefulness of the automatic condition.
Everything except the final questionnaire is logged inside the app. The Balanced Eater app already consisted of an exhaustive logging mechanism, which logs everything the user does. The final questionnaire was conducted in a Google Forms[2], which is a service from Google to create online surveys. When the app is launched the first time, the participants are asked to fill in a demographic questionnaire entering their age, gender and experience with the Balanced Eater app. The participants' food pictures are uploaded to a self hosted database, as well as the results of the

2 Google Forms https://www.google.com/forms/about/

automatic food analysis.
Furthermore, the results of the NASA-TLX as well as the SUS-questionnaire are also collected in that database.

7.3 RESULTS

In the following section, the results of the examined study are presented. During the study days, a total amount of 1378 food items on 539 different images have been tracked. The report of the results is structured in the same order as presented in the research questions.

7.3.1 *Efficiency*

The new system is efficient if automatically classifying the food items is faster than selecting items by hand. It is measured by the time taken for the participants to perform the food classification. The time tracking starts when the participants first sees the classification screen and stops once they hit the "next" button on that screen, excluding the time it takes to send the picture to the server and the results back. The sending of the pictures takes an average of 3.4 seconds (M = 3.0, SD = 1.3). This time is excluded in the average time of the classification, as it could be drastically reduced when using the trained model locally on the smartphone compared to a server-client solution. The automatic analysis also includes the time the participants spend looking at the results of the segmentation (for example, if they enlarge the image to take a closer look at the results). This could mean that the actual time might be lower than depicted in Table 7.2.
To further investigate this behavior, Figure 7.2 plots the needed classification time per day. For the automatic classification, a learning effect for the participants can be seen. Except for day two, the needed mean time is getting lower every day. A reason for this might be, that the participants get less interested in the results. Interestingly, there is no peak visible on day five for the mean time. This would have been expected, as from day four to day five, the condition was changed. When looking at the median (see Figure 7.2 right), this peak is visible for the automatic condition. More interesting is the fact, that on day six the needed time for the automatic condition is drastically reduced. This shows, that the participants either learned the system, or the participants might not look that long at the results anymore. A longer study would be needed to better understand this phenomenon, but this study already indicates a time saving from the automatic condition compared to the manual condition.

Kind	Manual	Automatic
Mean	45.5s	42.5s
Standard Deviation	57.6s	53.0s

Table 7.2: Seconds needed for a tracking on average with the corresponding standard deviation.

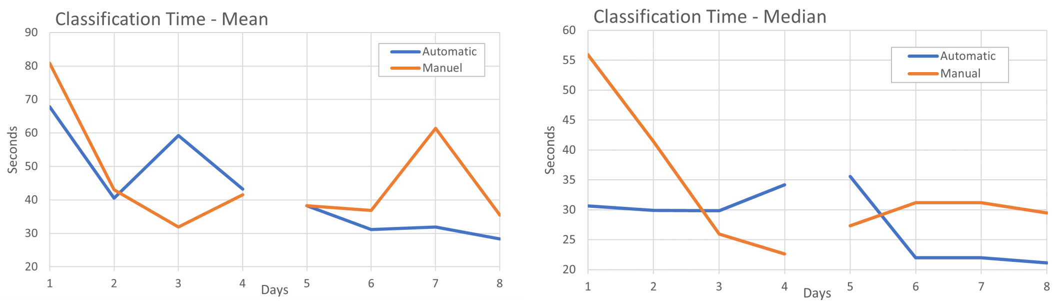


Figure 7.2: This figure shows the time it took the participants to track their food intake.

7.3.2 *Task-Load*

Through the NASA-TLX questionnaire, the subjects' load during the performance of the tasks is measured [25]. The questionnaire consists of six questions asking questions about the mental-, physical-, temporal- demand, the performance, the effort and the frustration of the participants. The participants rate each of the questions on a scale from zero to twenty. These values are then multiplied by five to get values from zero to one hundred. The values collected can be used to compare the task load of the two different conditions. The results are depicted in Figure 7.3 and show that the two conditions have similar values for most of the six categories. The results of the individual categories are presented in more detail in the following.

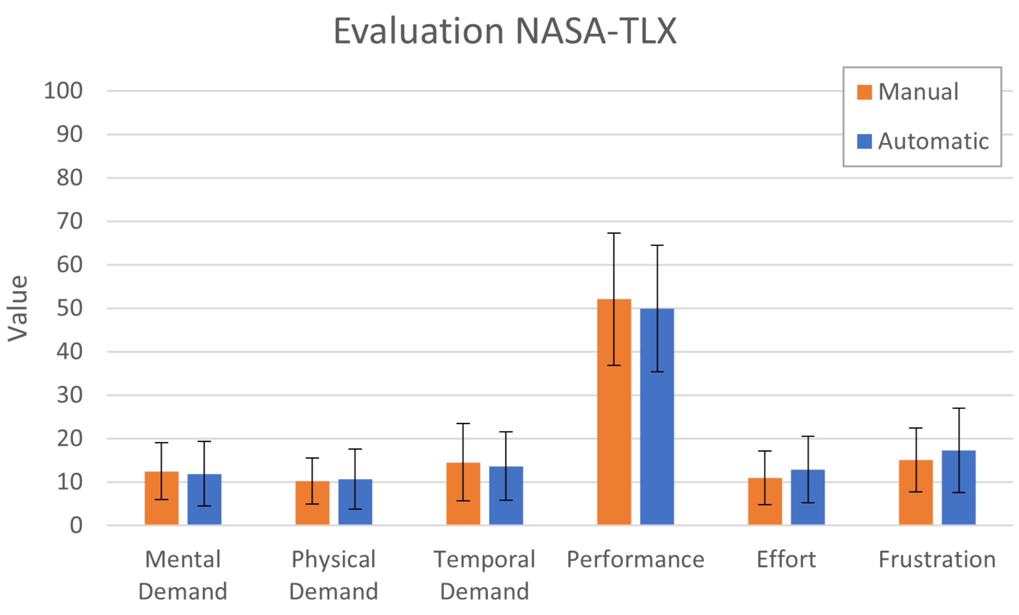


Figure 7.3: This figure shows the results of the NASA-TLX questionnaire.

7.3.2.1 *Mental Demand*

The mental demand category captures how mentally demanding the task was for the user. This is important because if the task is too mentally demanding, the users will not use the app for long. The hypothesis is that the automation of a process should make the task less mentally demanding (H2). According to the results this could not clearly be shown. The value of the automated condition is with 11.9 slightly better then the value for the The manual condition with 12.5, but the values are too close to see a clear trend (see Table 7.3).

Kind	Manual	Automatic
Mean	12.5	11.9
Standard Deviation	6.6	9.4

Table 7.3: Results of the NASA-TLX questionnaire for the category "Mental Demand"

7.3.2.2 *Physical Demand*

The physical demand category tracks how physically exhausting the task is. As the tested system was an app where the users track their food-intake by adding different categories, the system's physical demand was expected to be low. The manual condition reached a value of 10.2, while the automatic condition reached a value of 10.7. That means that the automatic system was physically more demanding for the participants than the manual condition. Although the values are close together, the gap could be explained by the fact that the participants need to review the results in the automatic classification and then correct it if it is wrong. Nevertheless, these values need to be considered with caution, as the feedback indicated that the participants did not understand this question, as they did not feel physically demanded at all.

Kind	Manual	Automatic
Mean	10.2	10.7
Standard Deviation	5.2	8.8

Table 7.4: Results of the NASA-TLX questionnaire for the category "Physical Demand"

7.3.2.3 *Temporal Demand*

The temporal demand category is essential for this thesis, as the primary goal was to reduce the time needed to track the food-intake. Hence, with this question the perceived time spend is tracked which does not necessarily need to align with the actual time needed to perform the task. But still the user should feel a time improvement compared to manual tracking. Table 7.5 shows that this goal could be achieved, although the values are close together. While the manual condition achieved a value of 14.6, the automatic condition achieved a value of 13.7. This shows that the participants had the feeling that they needed less time when using the automated condition. While this difference is small, it shows potential and could be extended if the classification accuracy can be improved.

Kind	Manual	Automatic
Mean	14.6	13.7
Standard Deviation	12.2	10.0

Table 7.5: Results of the NASA-TLX questionnaire for the category "Temporal Demand"

7.3.2.4 *Performance*

The performance category catches the subjective performance measure of the participants. The participants rated how well they feel, that they could achieve their goal with the different conditions. To keep it in line with the other questions, a lower score means better performance. The participants rated the manual condition with 52.1 compared to 50.0 for the automatic condition (see Table 7.6). Compared to the other values of the questionnaire, these two values are in general high. This shows that the participants, in general, had problems using the system to reach their goals. A reason for this could be that the system consisted of many questionnaires, which slowed down the actual task of tracking their food intake.

Kind	Manual	Automatic
Mean	52.1	50.0
Standard Deviation	23.4	25.3

Table 7.6: Results of the NASA-TLX questionnaire for the category "Performance"

7.3.2.5 *Effort*

The effort category captures how much effort the participants need to invest in getting to their goal. The manual condition performs with a value of 10.9 better than the automatic condition with a score of 12.9. The expectation was that the automation of the process could reduce the effort (H2). A reason for this could be that the deletion and adding of food items cause more effort than adding the food items from the start.

Kind	Manual	Automatic
Mean	10.9	12.9
Standard Deviation	7.3	9.4

Table 7.7: Results of the NASA-TLX questionnaire for the category "Effort"

7.3.2.6 *Frustration*

The frustration category tracks how frustrated the users are when using the system. This is important as if the system causes much frustration, the users might not use it, even if the system's performance is better than the other system. As Table 7.8 shows, the value of the frustration is with 17.3 higher than the value of the manual condition with 15.1. The higher standard deviation of the automatic condition could indicate that it is frustrating for the users if the system does not or detect wrong food items.

Kind	Manual	Automatic
Mean	15.1	17.3
Standard Deviation	7.4	12.5

Table 7.8: Results of the NASA-TLX questionnaire for the category "Frustration"

7.3.3 *Usability*

The SUS-questionnaire evaluates the usability of the system. The results are presented in Table 7.9. The SUS-questionnaire consists of ten standardized statements. The individual statements are rated on a scale from zero to four. The questionnaire is balanced, and therefore, every second question is negated. This means that a high value is good for uneven questions, while for even questions, a low value is good. The scores of the individual questions is then summed up and multiplied by 2.5. The resulting value is between zero and one hundred. Bangor et al. [2] introduced a rating scale for the SUS-questionnaire which shows that systems within a range of 52 to 73 indicate an okay usability. Everything above a score of 73 is very good, and everything below a score of 52 indicates bad usability.
When looking at the first questions' results, which checks if the participants could imagine using the system frequently, it shows that both conditions scored poorly. The reason might be that the app includes many questionnaires that need to be filled out to conduct a study, and therefore the app is not designed to be used in everyday life. Though, interestingly, the automatic condition is rated worse. For the questions four, five, and seven the difference between the two conditions is the highest and all indicate, that the automatic condition is easier to use and can be learned faster. Overall, the score of the automatic condition is higher than the score of the manual condition. This indicates an improvement in usability.
When computing the summed values for this scale, the manual condition reaches a value of 68.75, while the automatic condition reaches a score of 71.61 (see Figure 7.4). Therefore, the automatic condition improves the system's usability, but both systems are in an okay to good usability range.
As the app is a study application and not an application designed for everyday-life, these scores are acceptable and could probably be improved when reducing questionnaires.

7.3.4 *Quality*

To assess the quality of the automatic process of segmentation and classification, two different approaches were used: a subjective rating of the participants and an objective performance test. First of all, the participants rated their subjective feeling

Statement	Manual	Automatic
1. I think that I would like to use this system frequently.	1.33	1.14
2. I found the system unnecessarily complex.	0.80	1.00
3. I thought the system was easy to use.	2.93	3.14
4. I think that I would need the support of a technical person to be able to use this system.	0.67	0.29
5. I found the various functions in this system were well integrated.	2.40	2.79
6. I thought there was too much inconsistency in this system.	1.33	1.14
7. I would imagine that most people would learn to use this system very quickly.	2.67	3.21
8. I found the system very cumbersome to use.	1.40	1.36
9. I felt very confident using the system.	2.53	2.79
10. I needed to learn a lot of things before I could get going with this system.	0.87	0.64

Table 7.9: Results of the SUS-questionnaire. The scale goes from zero to four, while zero means strongly disagree and four means strongly agree.

of the quality after every food tracking and gave a final feedback on their perception. As a second step, the model was tested on a, for the model unseen, subset of the dataset to get an objective performance measure of the model.
After each meal tracking, the quality was tracked on a slider, which ranged from zero to one hundred, where zero means a bad result and one hundred means a good result. The classification reached a score of 60 (M = 63, SD = 35) and the segmentation a score of 54 (M = 54, SD =34). The objective performance measure also resulted in an accuracy of 61%. Therefore, the network detects almost two-third of the images correctly.
There were also two open questions in the final questionnaire asking the participants about feedback for the classification and segmentation. The feedback was relatively positive and showed that the model is excellent in detecting single objects or well-separated meals but is not that good for mixed meals.
Another created statistic is the amount of added and removed food items (see Figure 7.5). This statistics help to understand the quality of the classification. If the participants need to add a lot of classes, then the detection is not good. Even worse would be, if the participants need to remove a lot of classes. This would mean, that the classification classifies images incorrectly, which leads to additional work for the participants. On average, the participants added 3.14 food items with a total of 629 food items when using the manual condition and 1.72 food items with a total of 383 food items when using the automatic condition. However, the average of removed food items increased from 0.155 with a total of 31 food items for the manual con-

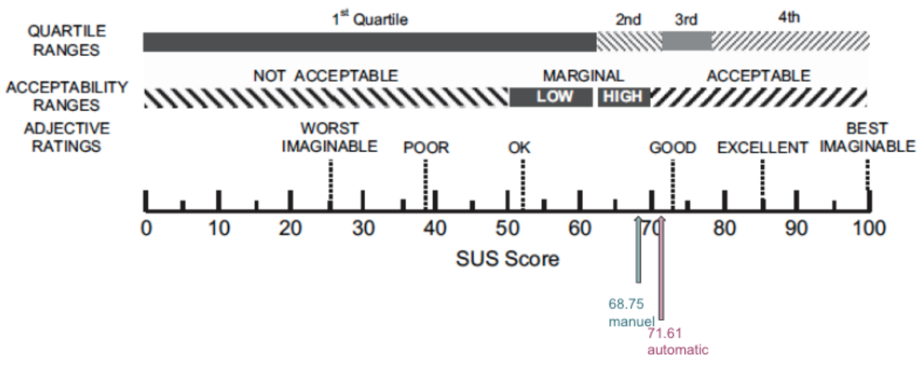


Figure 7.4: This figure shows the scale developed by Bangor et al. [2] and should help to classify different results. For better comparison negated question where converted.

dition to 0.55 food items with a total of 122 food items for the automatic condition.

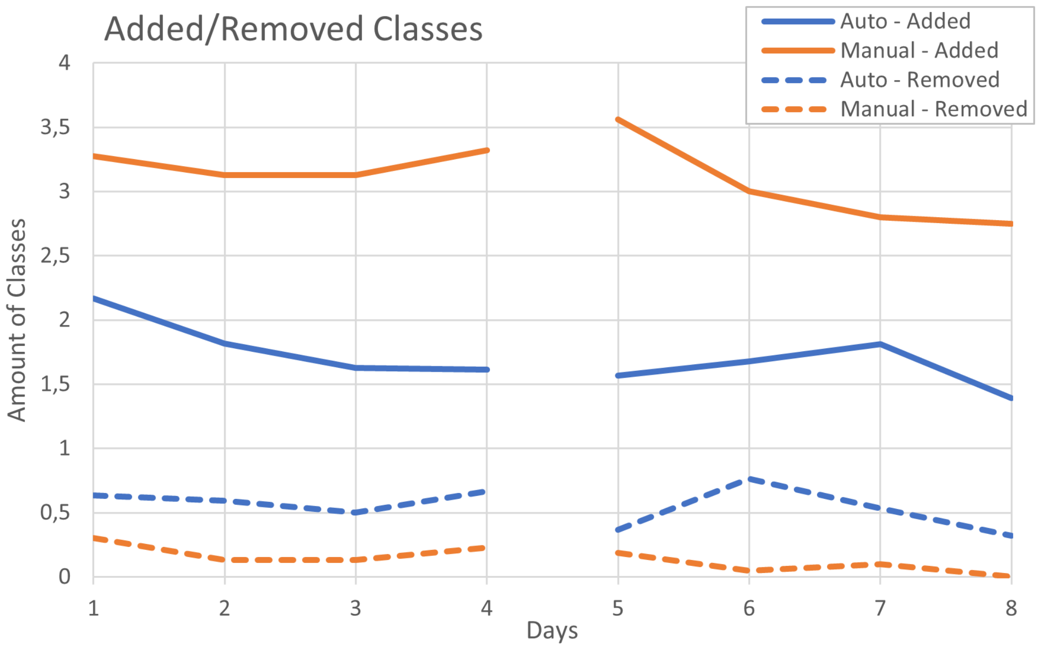


Figure 7.5: This figure shows the amount of times food items were added or removed for the two different conditions.

7.4 DISCUSSION

In this section the previous presented results will be used to answer and discuss the research questions introduced in the beginning of the chapter. The following summarizes the findings for each research questions. Furthermore, the most interesting findings are collected in a box after the summary.

7.4.1 *Efficiency*

On average, the automatic condition could save around three seconds compared to the manual condition. However, this amount equals more or less the time needed to send the image to the server and wait for the results. This time could be reduced by evaluating the model on the mobile device itself rather than on the server. Additionally, the time logged for the automated condition includes the time the participants looked at the results of the classification and segmentation. Although the app did not record the time spent looking at the results, Figure 7.2 shows that the time needed, especially for the automated condition, is getting lower each passing day. The reason for this could be, that the participants are more interested in the results in the beginning, because they are fascinated of the automatic detection. Some of the participants also reported that they learned to arrange their meal so that the model will detect it more reliable. To answer the initial hypothesis (H1) the evaluation shows, that there is a potential speed up, but it is not clear how large this speed up is.

RQ 1. *Efficiency*

- *Classification time could be reduced*
- *The time saved is needed to send and receive the results from the server*
- *Additional time might be saved, depending on how long the participants looked at the results*

7.4.2 *Task Load*

For the task load, the results are two folded. On the one hand, the mental and temporal demand, as well as the performance, was slightly improved. This might indicate that the automation of the process reduces work for the participant. On the other hand, effort and frustration of the participants were increased. This is interesting, as the hypothesis was, that automating a process would reduce effort and frustration (H2). A reason for this might be that the accuracy of the model is not yet high enough so that the user has additional work while correcting the automatic produced output. Two solutions could improve that situation. First, it might be a good idea to only suggest the found food classes to the user instead of directly adding them. That way, the user could use the suggestions but do not have to remove the suggestions if they are wrong. This could speed up the food tracking process if the suggestions are correct and will not slow down the process if they are not. Another idea would be to suggest the classes only when the model is sure about the class to a very high value. This is already the case, and the model suggests only food items, where the calculated probability is above 80%. This value might be too low and could be increased. Furthermore, the results showed that the model is good at detecting separated meal parts. This means that it might be beneficial if the model only tries to classify food parts that do not overlap with other food parts.

RQ 2. *Task Load*

- *Temporal- and mental demand, as well as performance, was improved*
- *Physical demand, effort, and frustration was increased*

7.4.3 *Usability*

The automated condition scored higher SUS values in almost each of the questions. Although the difference is often not large, this shows the potential of the created system. However, the SUS questionnaire also shows that the participants could not imagine using the system frequently. This is expected, as the system includes many questionnaires and therefore slows down the actual food-tracking. Nevertheless, when looking at the system's classification in the usability scale of Bangor et al. [2], the system still indicates an almost good usability value and hence is okay for a study application.

RQ 3. *Usability*

- *Values were improved in almost all questions*
- *Participants can not imagine using the system frequently*
- *The automated condition is easy to learn, and not much technical support is needed*

7.4.4 *Quality*

The participants rate the classification quality with a value of 60 from 100, which also matches the objective performance value. These values mean that, on average, two out of three items were classified correctly. The positive feedback of the participants in the open questions might indicate that the quality is already high enough, but when looking at the higher frustration values revealed in the NASA-TLX, this value might still be to low.
The answer to the research question depends on the use and input of the model. Suppose the goal is to depend on the model's correctness, then the answer should be no. If the model is used in an application where suggestions of food classes are provided to the user, the answer should be yes. Additionally, the quality of the results depends on the type of the meals. If the meal consists of clearly separated parts, then the quality is good. If the meal consists of mixed components, then the quality gets worse.
Of course, the quality of the results also heavily depends on the quality of the training images received from MTurk. If the goal is to label every instance of grape, the training images should contain every instance of a grape, and therefore, the MTurk workers need to label each grape in the image.
Furthermore, the quality of the model could be improved by performing more intense model tuning. For this thesis, model tuning was only performed in a very limited manner and could be improved in future work.

After looking at the images gathered during the study, another very interesting insight could be generated. Although the same meals prepared by different participants could look very different, the same meals prepared by the same participants on different days look similar. That means that if the models were retrained on the corrected classes, the accuracy could be improved. This approach should already be feasible and should be investigated in further works.

RQ 4. *Quality*

- *Objective performance test reveals an accuracy of roughly 60%*
- *Feedback from participants is rather positive*
- *Single components and separated meal parts are detected well*
- *Composed meals are detected not as well*

8 SUMMARY & OUTLOOK

This work is split up in three parts: The first part exlpains the theory and related work of food image tracking. The second presents a pipeline, which can be used to get a Neural Network model as output given unlabeled training images as input. This model can then be used to detect and classify food parts in new images. Furthermore, this model was embedded into the existing BalancedEater application to automate the application's food tracking part. The third part evaluates the resulting application in a usability study to get insights about this application's performance and usability compared to the original application with manual food tracking. The pipeline requires four steps, which consists of (1) getting the data labeled, (2) extract and clean the results of the labeling, (3) train a Mask R-CNN model, and (4) finally deploy the trained model to a server, which serves the BalancedEater application (see Figure 8.1). To label the data, an Amazon MTurk template was utilized (1). Although using these templates saves a lot of implementation time and provides superior user interfaces, it has a few drawbacks. Compared to a custom job, one cannot quickly assess the workers' quality in real-time while working on the job but needs to clean and assess the submitted results afterward. Therefore, a KNIME workflow (2) was built, which performs this quality assessment. Simple metrics, like the existence of more than one food instance in the images, were implemented to reduce the large amount of 34.000 images. For the rest of the images, a manual check was performed to get a better quality of the segmentation. Once the data is cleaned, it is essential to save the images in a format, which the Mask R-CNN framework can use later on—for this work, saving each occurrence of food into a separate file and for each image creating a separate folder. This structure makes it easy for the framework to read in the ground truth values and train the network. With this done, the actual training of the network (3) happens using the framework of the Matterport project. This framework provides some examples and hence makes it easy to adapt to custom needs. The training outcome is a Mask R-CNN network model deployed to a server, which can then be used to communicate with the improved, adapted BalancedEater application (4). With this application, the user can take pictures and upload them to the server. The server will process the incoming images and transmits back the segmented results. Afterward, the users can remove the wrong food classes, add the correct ones, and send the resulting image back to the server. With these new images on the server, the model could be retrained and improved over time.

The resulting application was then evaluated in a usability study with fifteen participants, which lasted for eight days. The study's main goal was to investigate if the automatic condition could speed up the food tracking and still keep or even improve the usability of the system. Furthermore, the study should reveal if the classification and segmentation accuracy is high enough to be used in a real-world

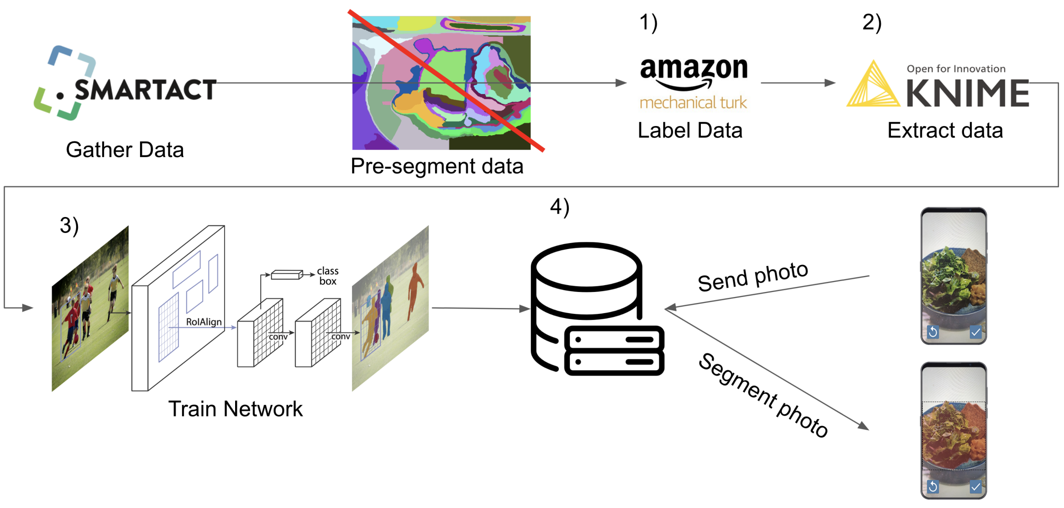


Figure 8.1: Shows the complete pipeline. The pre-segmentation was only used in the first draft of the pipeline, as it was not beneficial when an instance segmentation was the goal.

application.
The tracking time could be reduced when using the automatic condition. As it is unknown how long the participants looked at the segmentation results, the actual time reduction may be higher. The NASA-TLX showed that the automatic classification reduces temporal and mental demand and improves performance but increases physical demand, effort, and frustration. Interesting is the increase in frustration and effort. This might indicate that the system's quality is not high enough, and therefore removing wrongly added classes and adding the correct classes is more annoying for the participants than directly entering them themselves.
The usability of the automatic condition was overall rated better than for the manual condition. Both systems scored a bad value for the question if the participants could imagine using the system frequently. As the application is designed for studies and includes a lot of questionnaires, this is expected.

The results of the quality assessment of the system were two folded. On the one hand, the participants' mostly positive feedback showed the system's high potential and revealed that, especially for separated food parts, the system works well. On the other hand, the increase in the frustration and effort category of the NASA-TLX and the participants' feedback that the detection of mixed food parts does not work indicates that the quality is not high enough. The evaluation of the model on unseen images revealed an accuracy of about 60%. This shows that the system's quality might not be high enough for a purely automatic approach, but it could already give suggestions to the user.

Although the system's evaluation shows that the its accuracy is not high enough to be used in a purely automatic system, it also shows the huge potential the automation offers. The system can save the participants some time and provide information on the food items' position rather than only the food classes. This can be beneficial, for example, in the work of J. Wendler [72]. In her work, she analyzed the color of

food parts. The theory behind that was that the more colorful the food parts are, the healthier the intake is. To get the colors of the food parts, she used a mean-shift algorithm to segment the image and asked the users to select the image parts containing food. Although it worked, the segmentation was often not perfect, meaning the segments still contained some background parts. Furthermore, the pre-segmentation algorithm took some time to analyze the image. This processing time and the segments' accuracy could be improved using the created model of this work.
Another area the segmentation could be beneficial is when one wants to calculate the meal parts' volume. As all of the food images include a one euro coin, this coin could be used as a reference object, and the volume of the individual food parts could be calculated according to the size of the segments.
It could also be interesting to try to make the food classes more specific and therefore use a larger set of classes than the 25 generic classes used in this work. As the number of potential food classes is enormous and the amount of training images per class is often limited and expensive to gather, it is essential to balance the number of classes one wants to achieve and the number of required training images that one needs.
To improve the accuracy of the network, there are two main ideas. The first one is to perform further network-tuning. Not much parameter tuning was performed for this work, and only the network heads were retrained with the created dataset. There are still many different possibilities to tune the network and, therefore, improve the network's accuracy. Another approach would be to make the network further learn on the corrected images of the participants. One major problem with food images is that the same meals' food images can look very different. Although this is true for different participants, it turns out that the same participants prepare their meals very similar each time (see Figure 8.2). This knowledge can be used to retrain the network on the gathered images of the participants. Therefore, the network could be retrained, for example, once a week, and learn the participants' different individual food preparations.
Although there are still some obstacles to overcome, this thesis sets the ground for automatic food classification and segmentation and can be used as a basis for future research.

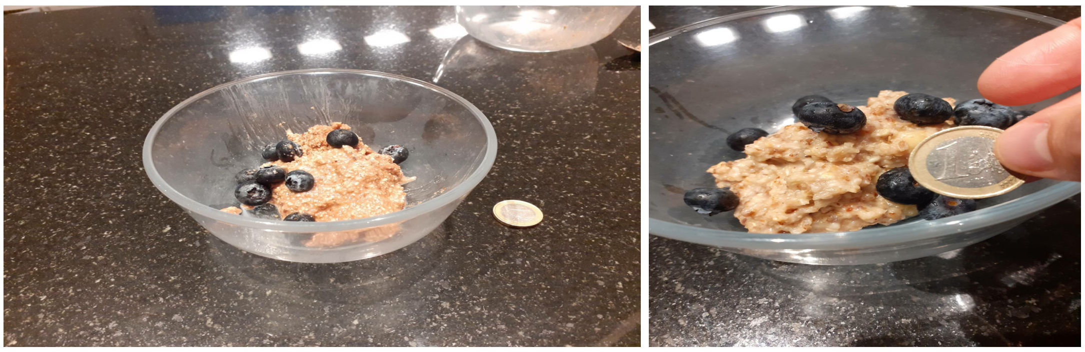


Figure 8.2: The figure shows the same meal for one participant at two different days. Although another participant might prepare that meal different, for the same participants it looks often very similar.

BIBLIOGRAPHY

[1] Anna-Lena. *Saftiger Apfelkuchen mit Streusel*. [Online; accessed December 5, 2020]. Year: Unkown. URL: https://www.einfachbacken.de/rezepte/apfelkuchen-mit-streusel-nach-omas-rezept.

[2] Aaron Bangor, Philip T. Kortum, and James T. Miller. "An Empirical Evaluation of the System Usability Scale". In: *International Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594. DOI: 10.1080/10447310802205776. eprint: https://doi.org/10.1080/10447310802205776. URL: https://doi.org/10.1080/10447310802205776.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. *Greedy Layer-Wise Training of Deep Networks*. Tech. rep. DOI: 10.5555/2976456.2976476.

[4] Bogenrieder. *Segmentation and Classification of real life foodimages*. Tech. rep. 2020.

[5] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. *Food-101-Mining Discriminative Components with Random Forests*. Tech. rep.

[6] Mei Chen, Kapil Dhingra, Wen Wu, Lei Yang, Rahul Sukthankar, and Jie Yang. *PFID: PITTSBURGH FAST-FOOD IMAGE DATASET*. Tech. rep. URL: http://pfid.intel-research.net.

[7] Christopher B Choy, Danfei Xu, Junyoung Gwak, Kevin Chen, and Silvio Savarese. *3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction*. Tech. rep. arXiv: 1604.00449v1.

[8] Gianluigi Ciocca, Paolo Napoletano, and Raimondo Schettini. "Food Recognition and Leftover Estimation for Daily Diet Monitoring". In: *New Trends in Image Analysis and Processing – ICIAP 2015 Workshops*. Ed. by Vittorio Murino, Enrico Puppo, Diego Sona, Marco Cristani, and Carlo Sansone. Vol. 9281. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 334–341. DOI: 10.1007/978-3-319-23222-5_41.

[9] Gianluigi Ciocca, Paolo Napoletano, and Raimondo Schettini. "Food recognition: a new dataset, experiments and results". In: *IEEE Journal of Biomedical and Health Informatics* 21.3 (2017), pp. 588–598. DOI: 10.1109/JBHI.2016.2636441.

[10] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/convolutional-networks/ (visited on Apr. 22, 2020).

[11] Le Cun, Jackel Henderson, Y Le Cun, J S Denker, D Henderson, R E Howard, W Hubbard, and L D Jackel. *Handwritten Digit Recognition with a Back-Propagation Network*. Tech. rep. 1990, pp. 396–404.

[12] Navneet Dalal and Bill Triggs. *Histograms of Oriented Gradients for Human Detection*. Tech. rep. URL: http://lear.inrialpes.fr.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. *ImageNet: A Large-Scale Hierarchical Image Database*. Tech. rep. URL: http://www.image-net.org..

[14] D. Eigen and R. Fergus. "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2650–2658.

[15] Mark Everingham, S. M.Ali Eslami, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (June 2014), pp. 98–136. ISSN: 15731405. DOI: 10.1007/s11263-014-0733-5.

[16] Theodoros Evgeniou and Massimiliano Pontil. *Support Vector Machines: Theory and Applications*. Vol. 2049 LNAI. 2001, pp. 249–257. DOI: 10.1007/3-540-44673-7_12. URL: http://link.springer.com/10.1007/3-540-44673-7%7B%5C_%7D12.

[17] Haoqiang Fan, Hao Su, and Leonidas Guibas. *A Point Set Generation Network for 3D Object Reconstruction from a Single Image*. Tech. rep.

[18] Shaobo Fang, Zeman Shao, Runyu Mao, Chichen Fu, Deborah A Kerr, Carol J Boushey, Edward J Delp, and Fengqing Zhu. *SINGLE-VIEW FOOD PORTION ESTIMATION: LEARNING IMAGE-TO-ENERGY MAPPINGS USING GENERATIVE ADVERSARIAL NETWORKS*. Tech. rep. 2018. arXiv: 1802.09670v2. URL: www.tadaproject.org..

[19] Luis C. García-Peraza-Herrera, Wenqi Li, Caspar Gruijthuijsen, Alain Devreker, George Attilakos, Jan Deprest, Emmanuel Vander Poorten, Danail Stoyanov, Tom Vercauteren, and Sébastien Ourselin. "Real-Time Segmentation of Non-rigid Surgical Tools Based on Deep Learning and Tracking". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10170 LNCS. Springer Verlag, 2017, pp. 84–95. DOI: 10.1007/978-3-319-54057-3_8. URL: http://link.springer.com/10.1007/978-3-319-54057-3%7B%5C_%7D8.

[20] Ross Girshick. *Fast R-CNN*. Tech. rep. arXiv: 1504.08083v2. URL: https://github.com/rbgirshick/.

[21] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Sept. 2014, pp. 580–587. ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524.

[22] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Nets*. Tech. rep. URL: http://www.github.com/goodfeli/adversarial.

[23] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings* (Dec. 2013). arXiv: 1312.6082. URL: http://arxiv.org/abs/1312.6082.

[24] Gutekueche.de. *Gedeckter Apfelkuchen aus Mürbteig*. [Online; accessed December 5, 2020]. Unkown. URL: https://www.gutekueche.de/gedeckter-apfelkuchen-aus-muerbteig-rezept-8623.

[25] Sandra G. Hart and Lowell E. Staveland. "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research". In: *Human Mental Workload*. Ed. by Peter A. Hancock and Najmedin Meshkati. Vol. 52. Advances in Psychology. North-Holland, 1988, pp. 139–183. DOI: https://doi.org/10.1016/S0166-4115(08)62386-9. URL: http://www.sciencedirect.com/science/article/pii/S0166411508623869.

[26] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (Feb. 2017), pp. 386–397. ISSN: 19393539. DOI: 10.1109/TPAMI.2018.2844175. arXiv: 1703.06870. URL: https://github.com/.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. Tech. rep. arXiv: 1512.03385v1. URL: http://image-net.org/challenges/LSVRC/2015/.

[28] Paul Henderson and Vittorio Ferrari. "Learning Single-Image 3D Reconstruction by Generative Modelling of Shape, Pose and Shading". In: *International Journal of Computer Vision* 128.4 (Apr. 2020), pp. 835–854. ISSN: 15731405. DOI: 10.1007/s11263-019-01219-8. arXiv: 1901.06447.

[29] Elnaz J. Heravi, Hamed H. Aghdam, and Domenec Puig. "Classification of foods by transferring knowledge from ImageNet dataset". In: *Ninth International Conference on Machine Vision (ICMV 2016)*. Ed. by Antanas Verikas, Petia Radeva, Dmitry P. Nikolaev, Wei Zhang, and Jianhong Zhou. Vol. 10341. SPIE, Mar. 2017, p. 1034128. DOI: 10.1117/12.2268737. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2268737.

[30] G E Hinton, N Srivastava, A Krizhevsky, I Sutskever, and R R Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors*. Tech. rep. arXiv: 1207.0580v1.

[31] Dimitris Iakovidis, Theodosis Goudas, Christos Smailis, and Ilias Maglogiannis. "Ratsnake: A Versatile Image Annotation Tool with Application to Computer-Aided Diagnosis". In: *TheScientificWorldJournal* 2014 (Jan. 2014), p. 286856. DOI: 10.1155/2014/286856.

[32] Ian Goodfellow, Yoshua Bengio and Aaron Courville. "Deep lLearning". In: *Genetic Programming and Evolvable Machines* 19.1-2 (2017), pp. 305–307. ISSN: 1389-2576. DOI: 10.1007/s10710-017-9314-z.

[33] *ImageNet*. [Online; accessed May 23, 2020]. URL: (http://image-net.org/about-overview.

[34] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. *The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation*. Tech. rep. arXiv: 1611.09326v3. URL: https://github.com/SimJeg/FC-DenseNet.

[35] A Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. Tech. rep. 2016. URL: https://cs231n.github.io/neural-networks-1/ (visited on Apr. 17, 2020).

[36] Hirokatsu Kataoka, Kenji Iwata, and Yutaka Satoh. "Feature Evaluation of Deep Convolutional Neural Networks for Object Recognition and Detection". In: (Sept. 2015). arXiv: 1509.07627. URL: http://arxiv.org/abs/1509.07627.

[37] Y. Kawano and K. Yanai. "Automatic Expansion of a Food Image Dataset Leveraging Existing Categories with Domain Adaptation". In: *Proc. of ECCV Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV)*. 2014.

[38] Yoshiyuki Kawano and Keiji Yanai. "FoodCam: A real-time mobile food recognition system employing Fisher Vector". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8326 LNCS. PART 2. Springer, Cham, 2014, pp. 369–373. ISBN: 9783319041162. DOI: 10.1007/978-3-319-04117-9_38.

[39] Yoshiyuki Kawano and Keiji Yanai. *Automatic Expansion of a Food Image Dataset Leveraging Existing Categories with Domain Adaptation*. Tech. rep.

[40] Laura M. König and Britta Renner. "Colourful = healthy? Exploring meal colour variety and its relation to food consumption". In: *Food Quality and Preference* 64 (Mar. 2018), pp. 66–71. ISSN: 09503293. DOI: 10.1016/j.foodqual.2017.10.011.

[41] Laura M. König and Britta Renner. "Boosting healthy food choices by meal colour variety: Results from two experiments and a just-in-time Ecological Momentary Intervention". In: *BMC Public Health* 19.1 (July 2019), p. 975. ISSN: 14712458. DOI: 10.1186/s12889-019-7306-z. URL: https://bmcpublichealth.biomedcentral.com/articles/10.1186/s12889-019-7306-z.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Tech. rep. URL: http://code.google.com/p/cuda-convnet/.

[43] Alina Kuznetsova et al. "The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale". In: *IJCV* (2020).

[44] Yanchao Liang and Jianhua Li. *Deep Learning-Based Food Calorie Estimation Method in Dietary Assessment $*. Tech. rep. arXiv: 1706.04062v4. URL: http://www.hc-sc.gc.ca/fn-an/nutrition/fiche-nutri-data/nutrient%7B%5C_%7D.

[45] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. Tech. rep. arXiv: 1312.4400v3.

[46] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft COCO: Common objects in context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8693 LNCS. PART 5. Springer Verlag, May 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48. arXiv: 1405.0312.

[47] David G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 09205691. DOI: 10.1023/B:VISI.0000029664.99615.94.

[48] Lucia. *Spaghetti a la Tofu-Bolognese*. [Online; accessed December 5, 2020]. 2020. URL: https://www.zentrum-der-gesundheit.de/rezepte/hauptgerichte/spaghetti-a-la-tofu-bolognese.

[49] Niki Martinel, Gian Luca Foresti, and Christian Micheloni. "Wide-Slice Residual Networks for Food Recognition". In: (Dec. 2016). arXiv: 1612.06543. URL: http://arxiv.org/abs/1612.06543.

[50] Y. Matsuda, H. Hoashi, and K. Yanai. "Recognition of Multiple-Food Images by Detecting Candidate Regions". In: *Proc. of IEEE International Conference on Multimedia and Expo (ICME)*. 2012.

[51] Simon Mezgec and Barbara Koroušić Seljak. "NutriNet: A Deep Learning Food and Drink Image Recognition System for Dietary Assessment." In: *Nutrients* 9.7 (June 2017). ISSN: 2072-6643. DOI: 10.3390/nu9070657. URL: http://www.ncbi.nlm.nih.gov/pubmed/28653995%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5537777.

[52] Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin Murphy. "Im2Calories: towards an automated mobile vision food diary". In: *ICCV*. 2015. URL: http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Meyers_Im2Calories_Towards_an_ICCV_2015_paper.pdf.

[53] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: (Nov. 2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[54] Kamila Poslusna, Jiri Ruprich, Jeanne H.M. De Vries, Marie Jakubikova, and Pieter Van'T Veer. "Misreporting of energy and micronutrient intake estimated by food records and 24hour recalls, control and adjustment methods in practice". In: *British Journal of Nutrition* 101.SUPPL. 2 (July 2009). ISSN: 00071145. DOI: 10.1017/S0007114509990602.

[55] Marcel Prastawa, Elizabeth Bullitt, Nathan Moon, Koen Van Leemput, and Guido Gerig. "Automatic brain tumor segmentation by subject specific modification of atlas priors." In: *Academic radiology* 10.12 (Dec. 2003), pp. 1341–8. ISSN: 1076-6332. DOI: 10.1016/s1076-6332(03)00506-3. URL: http://www.ncbi.nlm.nih.gov/pubmed/14697002%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2430604.

[56] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[57] F Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain.* Tech. rep. 6. 1958, pp. 19–27.

[58] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. *"GrabCut"-Interactive Foreground Extraction using Iterated Graph Cuts*. Tech. rep.

[59] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0.

[60] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. Tech. rep. arXiv: 1409.0575v3. URL: http://image-net.org/challenges/LSVRC/.

[61] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. *LabelMe: a database and web-based tool for image annotation*. Tech. rep. 3. 2008, pp. 157–173.

[62] S H Shabbeer Basha, Shiv Ram Dubey, Viswanath Pulabaigari, and Snehasis Mukherjee. “Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification”. In: (2019). DOI: 10.1016/j.neucom.2019.10.008. arXiv: 1902.02771v3. URL: https://doi.org/10.1016/j.neucom.2019.10.008.

[63] Tianxiang Shen, Ruixian Liu, Ju Bai, and Zheng Li. *"Deep Fakes" using Generative Adversarial Networks (GAN)*. Tech. rep.

[64] Wataru Shimoda and Keiji Yanai. “CNN-based food image segmentation without pixel-wise annotation”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9281. Springer Verlag, 2015, pp. 449–457. ISBN: 9783319232218. DOI: 10.1007/978-3-319-23222-5_55. URL: http://koen.me/research/selectivesearch/.

[65] Karen Simonyan and Andrew Zisserman. *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*. Tech. rep. 2015. arXiv: 1409.1556v6. URL: http://www.robots.ox.ac.uk/.

[66] SuperValu. *OHHH SO EASY SPAGHETTI BOLOGNESE*. [Online; accessed December 5, 2020]. Unkown. URL: https://supervalu.ie/real-food/recipes/ohhh-so-easy-spaghetti-bolognese.

[67] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. Tech. rep.

[68] Ryosuke Tanno, Koichi Okamoto, and Keiji Yanai. “DeepFoodCam: A DCNN-based real-time mobile food recognition system”. In: *MADiMa 2016 - Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management, co-located with ACM Multimedia 2016*. New York, New York, USA: Association for Computing Machinery, Inc, Oct. 2016, p. 89. ISBN: 9781450345200. DOI: 10.1145/2986035.2986044. URL: http://dl.acm.org/citation.cfm?doid=2986035.2986044.

[69] Martin Thoma. “A Survey of Semantic Segmentation”. In: (Feb. 2016). arXiv: 1602.06541. URL: http://arxiv.org/abs/1602.06541.

[70] Karoline Villinger, Deborah R. Wahl, Laura M. König, Katrin Ziesemer, Simon Butscher, Jens Müller, Harald Reiterer, Harald T. Schupp, and Britta Renner. "Do We Know What We Enjoy? Accuracy of Forecasted Eating Happiness". In: *Frontiers in Psychology* 11 (June 2020), p. 1187. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2020.01187. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2020.01187/full.

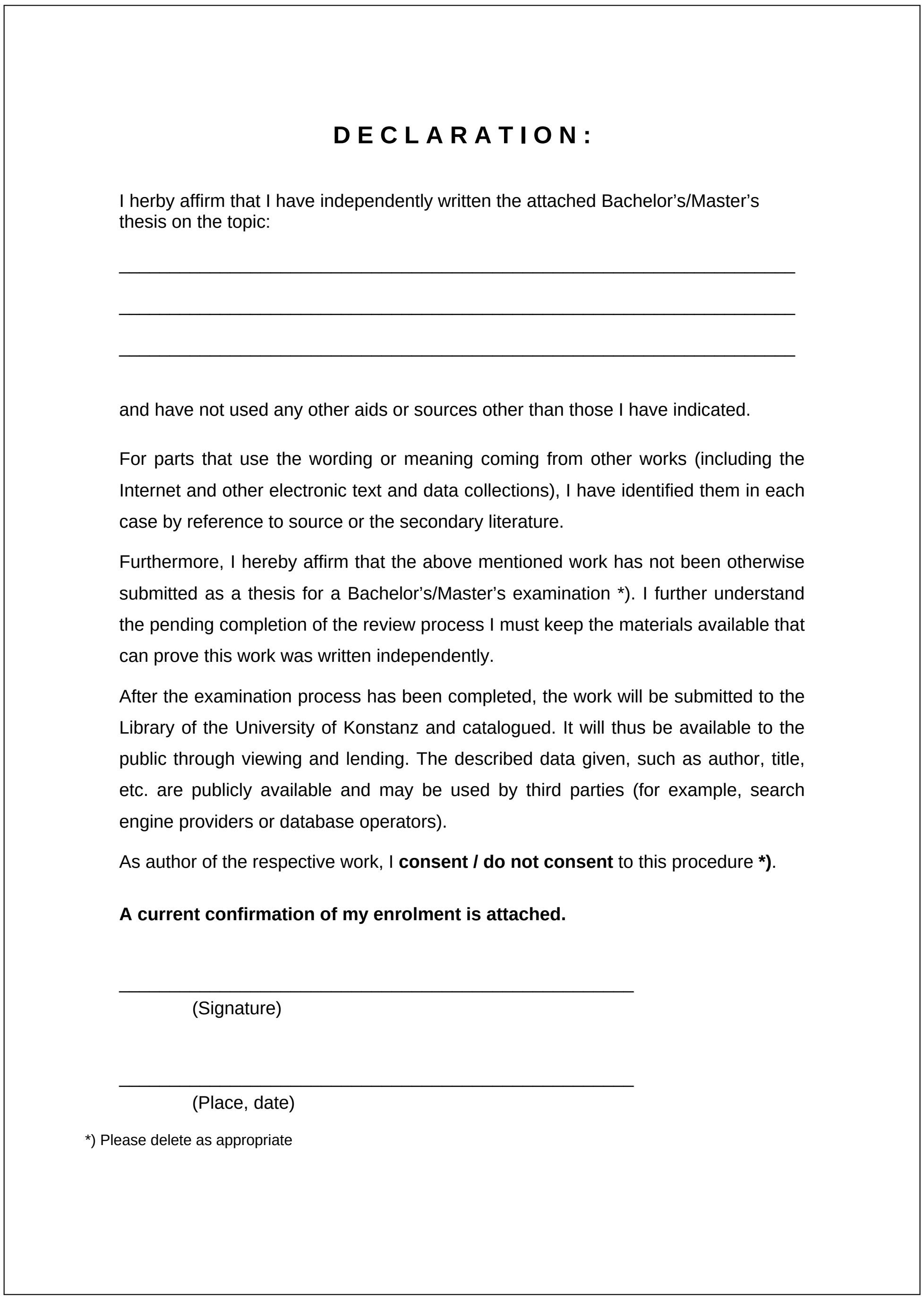
[71] Deborah Ronja Wahl, Karoline Villinger, Michael Blumenschein, Laura Maria König, Katrin Ziesemer, Gudrun Sproesser, Harald Thomas Schupp, and Britta Renner. "Why We Eat What We Eat: Assessing Dispositional and In-the-Moment Eating Motives by Using Ecological Momentary Assessment." In: *JMIR mHealth and uHealth* 8.1 (Jan. 2020), e13191. ISSN: 2291-5222. DOI: 10.2196/13191. URL: http://www.ncbi.nlm.nih.gov/pubmed/31909719%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6996745.

[72] Juliane Wendler. "Eat your Colors - Konzeption, Implementierung und Evaluation eines mobilen Ernährungstagebuches für das farbbasierte Erfassen der Ernährung". Bachelor Thesis. University of Konstanz, 2019.

[73] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. *Convolutional neural networks: an overview and application in radiology*. Aug. 2018. DOI: 10.1007/s13244-018-0639-9.

[74] Chuanhai Zhang, Kurt Loken, Zhiyu Chen, Zhiyong Xiao, and Gary Kunkel. *Mask Editor : an Image Annotation Tool for Image Segmentation Tasks*. Tech. rep. 2018.

A DECLARATION OF INDEPENDENT WORK



DECLARATION:

I herby affirm that I have independently written the attached Bachelor's/Master's thesis on the topic:

____

____

____

and have not used any other aids or sources other than those I have indicated.

For parts that use the wording or meaning coming from other works (including the Internet and other electronic text and data collections), I have identified them in each case by reference to source or the secondary literature.

Furthermore, I hereby affirm that the above mentioned work has not been otherwise submitted as a thesis for a Bachelor's/Master's examination *). I further understand the pending completion of the review process I must keep the materials available that can prove this work was written independently.

After the examination process has been completed, the work will be submitted to the Library of the University of Konstanz and catalogued. It will thus be available to the public through viewing and lending. The described data given, such as author, title, etc. are publicly available and may be used by third parties (for example, search engine providers or database operators).

As author of the respective work, I **consent / do not consent** to this procedure ***)**.

A current confirmation of my enrolment is attached.

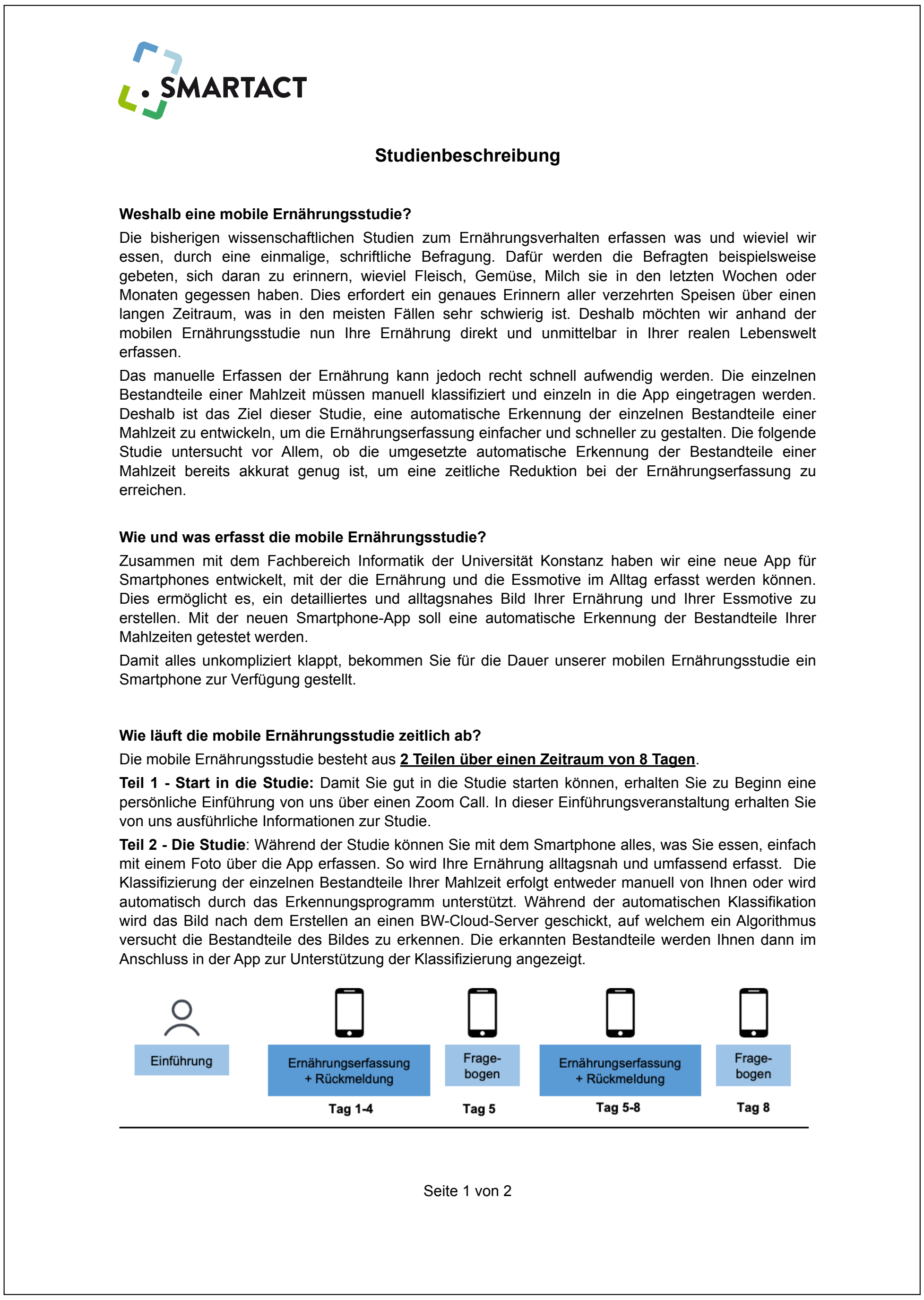
____
(Signature)

____
(Place, date)

*) Please delete as appropriate

Figure A.1: Declaration of independent work

B USABILITY STUDY MATERIAL



SMARTACT

Studienbeschreibung

Weshalb eine mobile Ernährungsstudie?

Die bisherigen wissenschaftlichen Studien zum Ernährungsverhalten erfassen was und wieviel wir essen, durch eine einmalige, schriftliche Befragung. Dafür werden die Befragten beispielsweise gebeten, sich daran zu erinnern, wieviel Fleisch, Gemüse, Milch sie in den letzten Wochen oder Monaten gegessen haben. Dies erfordert ein genaues Erinnern aller verzehrten Speisen über einen langen Zeitraum, was in den meisten Fällen sehr schwierig ist. Deshalb möchten wir anhand der mobilen Ernährungsstudie nun Ihre Ernährung direkt und unmittelbar in Ihrer realen Lebenswelt erfassen.

Das manuelle Erfassen der Ernährung kann jedoch recht schnell aufwendig werden. Die einzelnen Bestandteile einer Mahlzeit müssen manuell klassifiziert und einzeln in die App eingetragen werden. Deshalb ist das Ziel dieser Studie, eine automatische Erkennung der einzelnen Bestandteile einer Mahlzeit zu entwickeln, um die Ernährungserfassung einfacher und schneller zu gestalten. Die folgende Studie untersucht vor Allem, ob die umgesetzte automatische Erkennung der Bestandteile einer Mahlzeit bereits akkurat genug ist, um eine zeitliche Reduktion bei der Ernährungserfassung zu erreichen.

Wie und was erfasst die mobile Ernährungsstudie?

Zusammen mit dem Fachbereich Informatik der Universität Konstanz haben wir eine neue App für Smartphones entwickelt, mit der die Ernährung und die Essmotive im Alltag erfasst werden können. Dies ermöglicht es, ein detailliertes und alltagsnahes Bild Ihrer Ernährung und Ihrer Essmotive zu erstellen. Mit der neuen Smartphone-App soll eine automatische Erkennung der Bestandteile Ihrer Mahlzeiten getestet werden.

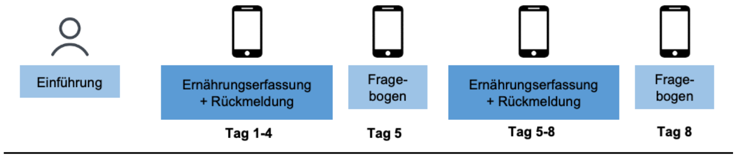
Damit alles unkompliziert klappt, bekommen Sie für die Dauer unserer mobilen Ernährungsstudie ein Smartphone zur Verfügung gestellt.

Wie läuft die mobile Ernährungsstudie zeitlich ab?

Die mobile Ernährungsstudie besteht aus **2 Teilen über einen Zeitraum von 8 Tagen**.

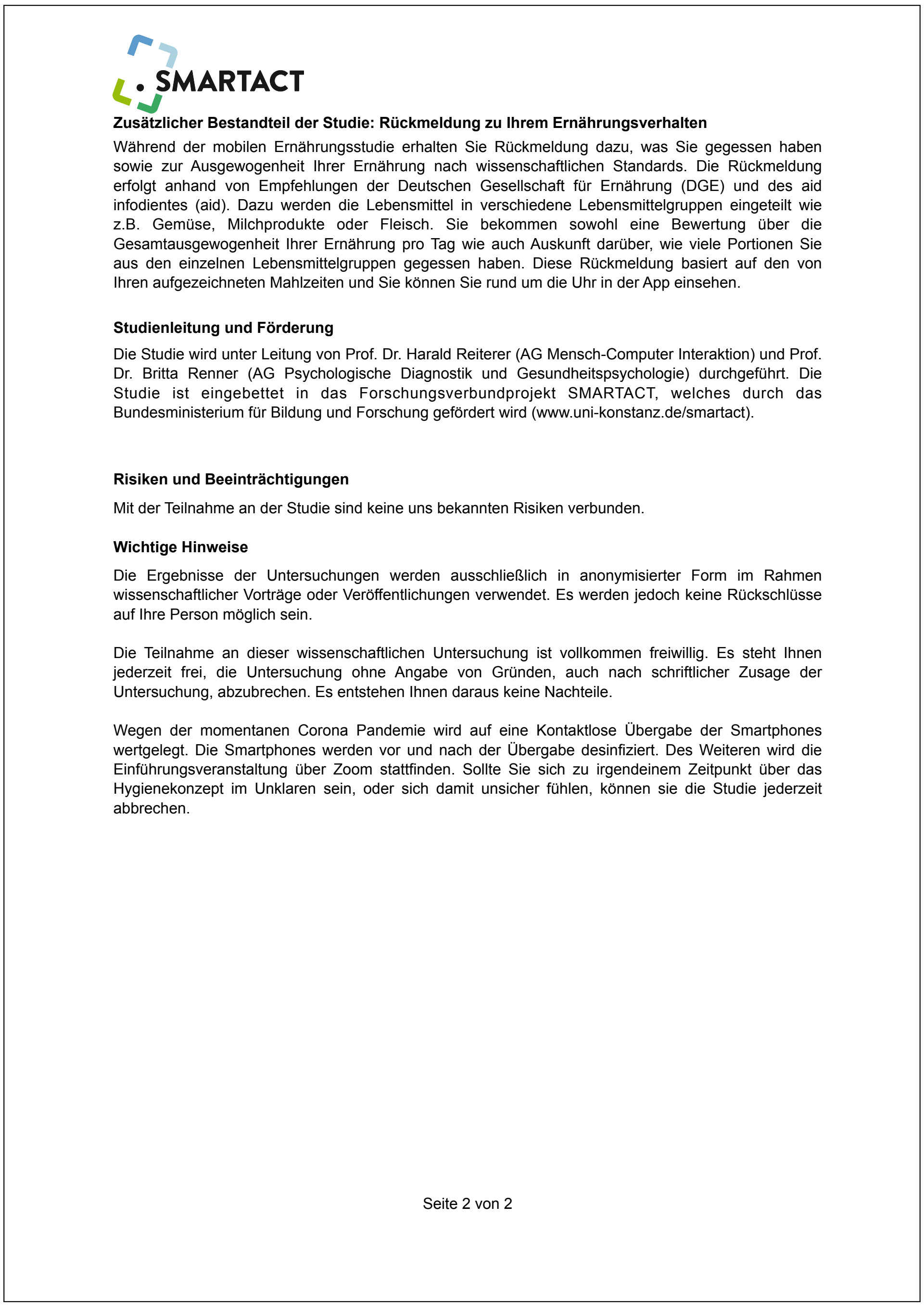
Teil 1 - Start in die Studie: Damit Sie gut in die Studie starten können, erhalten Sie zu Beginn eine persönliche Einführung von uns über einen Zoom Call. In dieser Einführungsveranstaltung erhalten Sie von uns ausführliche Informationen zur Studie.

Teil 2 - Die Studie: Während der Studie können Sie mit dem Smartphone alles, was Sie essen, einfach mit einem Foto über die App erfassen. So wird Ihre Ernährung alltagsnah und umfassend erfasst. Die Klassifizierung der einzelnen Bestandteile Ihrer Mahlzeit erfolgt entweder manuell von Ihnen oder wird automatisch durch das Erkennungsprogramm unterstützt. Während der automatischen Klassifikation wird das Bild nach dem Erstellen an einen BW-Cloud-Server geschickt, auf welchem ein Algorithmus versucht die Bestandteile des Bildes zu erkennen. Die erkannten Bestandteile werden Ihnen dann im Anschluss in der App zur Unterstützung der Klassifizierung angezeigt.


Einführung
Ernährungserfassung + Rückmeldung
Tag 1-4
Frage-bogen
Tag 5
Ernährungserfassung + Rückmeldung
Tag 5-8
Frage-bogen
Tag 8

Seite 1 von 2

Figure B.1: Description of the Study page 1



SMARTACT

Zusätzlicher Bestandteil der Studie: Rückmeldung zu Ihrem Ernährungsverhalten

Während der mobilen Ernährungsstudie erhalten Sie Rückmeldung dazu, was Sie gegessen haben sowie zur Ausgewogenheit Ihrer Ernährung nach wissenschaftlichen Standards. Die Rückmeldung erfolgt anhand von Empfehlungen der Deutschen Gesellschaft für Ernährung (DGE) und des aid infodientes (aid). Dazu werden die Lebensmittel in verschiedene Lebensmittelgruppen eingeteilt wie z.B. Gemüse, Milchprodukte oder Fleisch. Sie bekommen sowohl eine Bewertung über die Gesamtausgewogenheit Ihrer Ernährung pro Tag wie auch Auskunft darüber, wie viele Portionen Sie aus den einzelnen Lebensmittelgruppen gegessen haben. Diese Rückmeldung basiert auf den von Ihren aufgezeichneten Mahlzeiten und Sie können Sie rund um die Uhr in der App einsehen.

Studienleitung und Förderung

Die Studie wird unter Leitung von Prof. Dr. Harald Reiterer (AG Mensch-Computer Interaktion) und Prof. Dr. Britta Renner (AG Psychologische Diagnostik und Gesundheitspsychologie) durchgeführt. Die Studie ist eingebettet in das Forschungsverbundprojekt SMARTACT, welches durch das Bundesministerium für Bildung und Forschung gefördert wird (www.uni-konstanz.de/smartact).

Risiken und Beeinträchtigungen

Mit der Teilnahme an der Studie sind keine uns bekannten Risiken verbunden.

Wichtige Hinweise

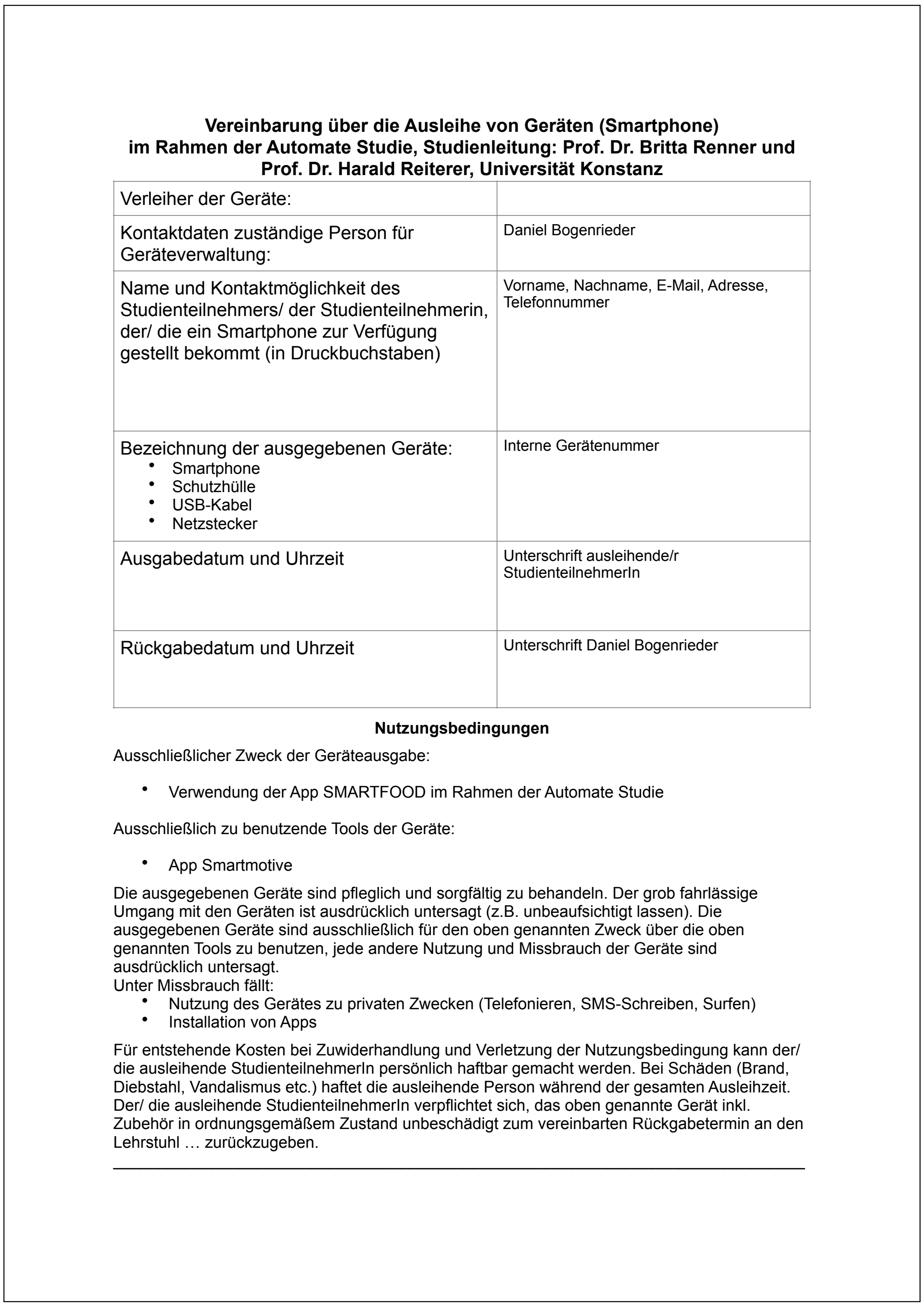
Die Ergebnisse der Untersuchungen werden ausschließlich in anonymisierter Form im Rahmen wissenschaftlicher Vorträge oder Veröffentlichungen verwendet. Es werden jedoch keine Rückschlüsse auf Ihre Person möglich sein.

Die Teilnahme an dieser wissenschaftlichen Untersuchung ist vollkommen freiwillig. Es steht Ihnen jederzeit frei, die Untersuchung ohne Angabe von Gründen, auch nach schriftlicher Zusage der Untersuchung, abzubrechen. Es entstehen Ihnen daraus keine Nachteile.

Wegen der momentanen Corona Pandemie wird auf eine Kontaktlose Übergabe der Smartphones wertgelegt. Die Smartphones werden vor und nach der Übergabe desinfiziert. Des Weiteren wird die Einführungsveranstaltung über Zoom stattfinden. Sollte Sie sich zu irgendeinem Zeitpunkt über das Hygienekonzept im Unklaren sein, oder sich damit unsicher fühlen, können sie die Studie jederzeit abbrechen.

Seite 2 von 2

Figure B.2: Description of the study page 2



Vereinbarung über die Ausleihe von Geräten (Smartphone) im Rahmen der Automate Studie, Studienleitung: Prof. Dr. Britta Renner und Prof. Dr. Harald Reiterer, Universität Konstanz

Verleiher der Geräte:	
Kontaktdaten zuständige Person für Geräteverwaltung:	Daniel Bogenrieder
Name und Kontaktmöglichkeit des Studienteilnehmers/ der Studienteilnehmerin, der/ die ein Smartphone zur Verfügung gestellt bekommt (in Druckbuchstaben)	Vorname, Nachname, E-Mail, Adresse, Telefonnummer
Bezeichnung der ausgegebenen Geräte: • Smartphone • Schutzhülle • USB-Kabel • Netzstecker	Interne Gerätenummer
Ausgabedatum und Uhrzeit	Unterschrift ausleihende/r StudienteilnehmerIn
Rückgabedatum und Uhrzeit	Unterschrift Daniel Bogenrieder

Nutzungsbedingungen

Ausschließlicher Zweck der Geräteausgabe:

- Verwendung der App SMARTFOOD im Rahmen der Automate Studie

Ausschließlich zu benutzende Tools der Geräte:

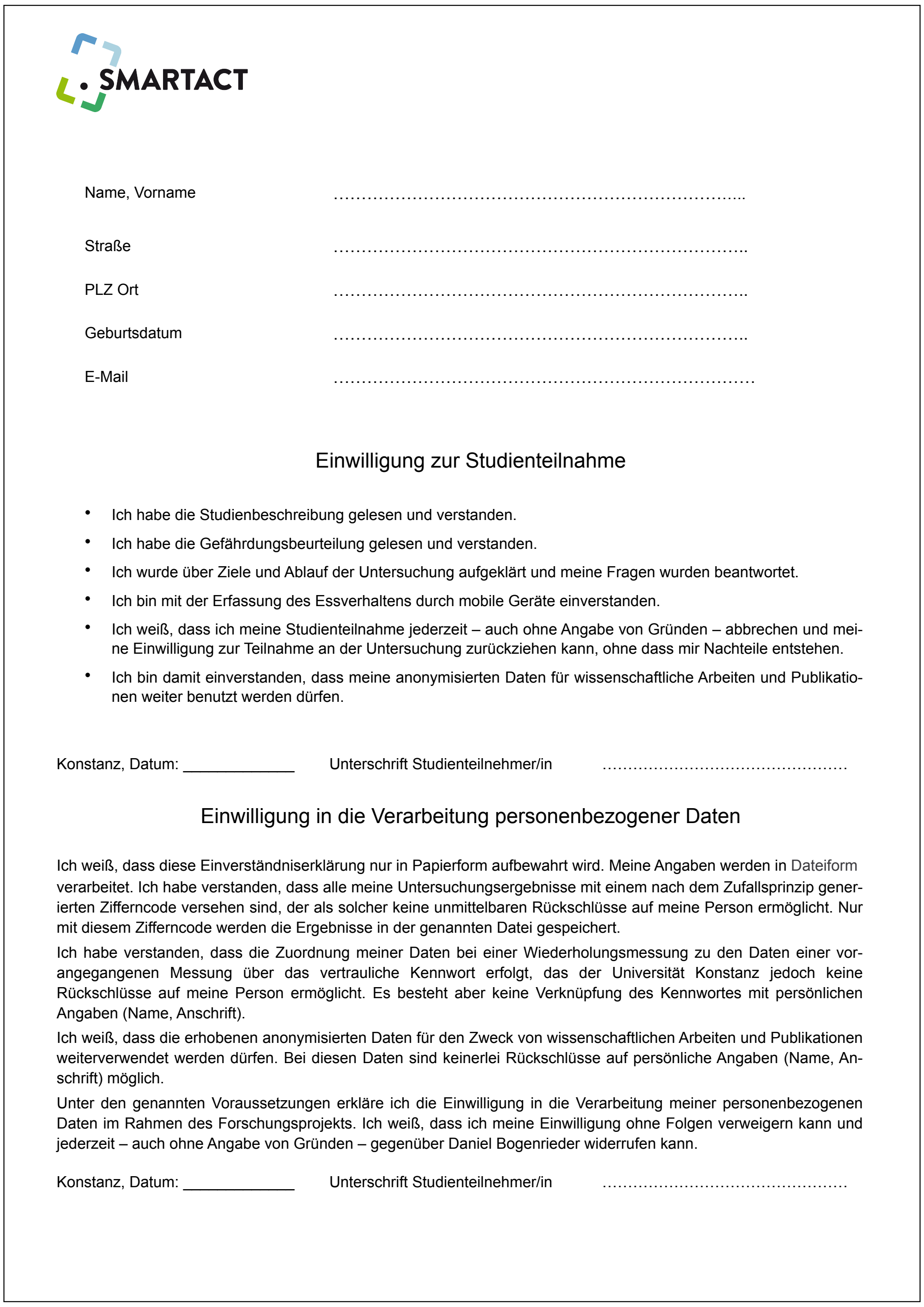
- App Smartmotive

Die ausgegebenen Geräte sind pfleglich und sorgfältig zu behandeln. Der grob fahrlässige Umgang mit den Geräten ist ausdrücklich untersagt (z.B. unbeaufsichtigt lassen). Die ausgegebenen Geräte sind ausschließlich für den oben genannten Zweck über die oben genannten Tools zu benutzen, jede andere Nutzung und Missbrauch der Geräte sind ausdrücklich untersagt.
Unter Missbrauch fällt:

- Nutzung des Gerätes zu privaten Zwecken (Telefonieren, SMS-Schreiben, Surfen)
- Installation von Apps

Für entstehende Kosten bei Zuwiderhandlung und Verletzung der Nutzungsbedingung kann der/ die ausleihende StudienteilnehmerIn persönlich haftbar gemacht werden. Bei Schäden (Brand, Diebstahl, Vandalismus etc.) haftet die ausleihende Person während der gesamten Ausleihzeit. Der/ die ausleihende StudienteilnehmerIn verpflichtet sich, das oben genannte Gerät inkl. Zubehör in ordnungsgemäßem Zustand unbeschädigt zum vereinbarten Rückgabetermin an den Lehrstuhl … zurückzugeben.

Figure B.3: Agreement on the loan of equipment



SMARTACT

Name, Vorname	………………………………………………………………...
Straße	………………………………………………………………..
PLZ Ort	………………………………………………………………..
Geburtsdatum	………………………………………………………………..
E-Mail	…………………………………………………………………

Einwilligung zur Studienteilnahme

- Ich habe die Studienbeschreibung gelesen und verstanden.
- Ich habe die Gefährdungsbeurteilung gelesen und verstanden.
- Ich wurde über Ziele und Ablauf der Untersuchung aufgeklärt und meine Fragen wurden beantwortet.
- Ich bin mit der Erfassung des Essverhaltens durch mobile Geräte einverstanden.
- Ich weiß, dass ich meine Studienteilnahme jederzeit – auch ohne Angabe von Gründen – abbrechen und meine Einwilligung zur Teilnahme an der Untersuchung zurückziehen kann, ohne dass mir Nachteile entstehen.
- Ich bin damit einverstanden, dass meine anonymisierten Daten für wissenschaftliche Arbeiten und Publikationen weiter benutzt werden dürfen.

Konstanz, Datum: _____________ Unterschrift Studienteilnehmer/in ……………………………………….

Einwilligung in die Verarbeitung personenbezogener Daten

Ich weiß, dass diese Einverständniserklärung nur in Papierform aufbewahrt wird. Meine Angaben werden in Dateiform verarbeitet. Ich habe verstanden, dass alle meine Untersuchungsergebnisse mit einem nach dem Zufallsprinzip generierten Zifferncode versehen sind, der als solcher keine unmittelbaren Rückschlüsse auf meine Person ermöglicht. Nur mit diesem Zifferncode werden die Ergebnisse in der genannten Datei gespeichert.

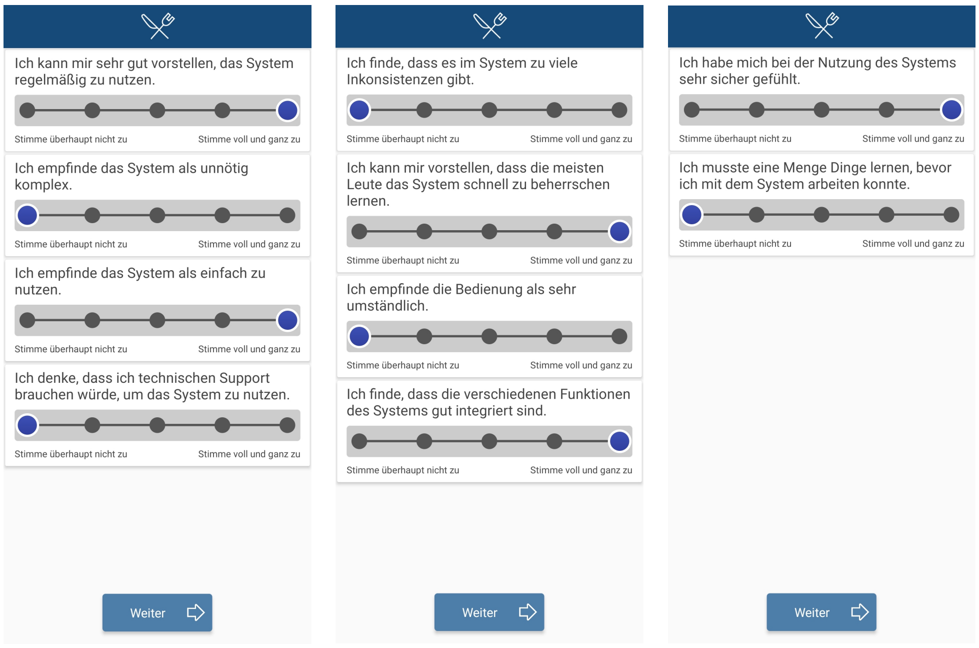
Ich habe verstanden, dass die Zuordnung meiner Daten bei einer Wiederholungsmessung zu den Daten einer vorangegangenen Messung über das vertrauliche Kennwort erfolgt, das der Universität Konstanz jedoch keine Rückschlüsse auf meine Person ermöglicht. Es besteht aber keine Verknüpfung des Kennwortes mit persönlichen Angaben (Name, Anschrift).

Ich weiß, dass die erhobenen anonymisierten Daten für den Zweck von wissenschaftlichen Arbeiten und Publikationen weiterverwendet werden dürfen. Bei diesen Daten sind keinerlei Rückschlüsse auf persönliche Angaben (Name, Anschrift) möglich.

Unter den genannten Voraussetzungen erkläre ich die Einwilligung in die Verarbeitung meiner personenbezogenen Daten im Rahmen des Forschungsprojekts. Ich weiß, dass ich meine Einwilligung ohne Folgen verweigern kann und jederzeit – auch ohne Angabe von Gründen – gegenüber Daniel Bogenrieder widerrufen kann.

Konstanz, Datum: _____________ Unterschrift Studienteilnehmer/in ……………………………………….

Figure B.4: Consent to study participation


Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.

● ● ● ● [✓ ●]

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Ich empfinde das System als unnötig komplex.

[✓ ●] ● ● ● ●

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Ich empfinde das System als einfach zu nutzen.

● ● ● ● [✓ ●]

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.

[✓ ●] ● ● ● ●

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Weiter

Ich finde, dass es im System zu viele Inkonsistenzen gibt.

[✓ ●] ● ● ● ●

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.

● ● ● ● [✓ ●]

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Ich empfinde die Bedienung als sehr umständlich.

[✓ ●] ● ● ● ●

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.

● ● ● ● [✓ ●]

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Weiter

Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.

● ● ● ● [✓ ●]

Stimme überhaupt nicht zu — Stimme voll und ganz zu

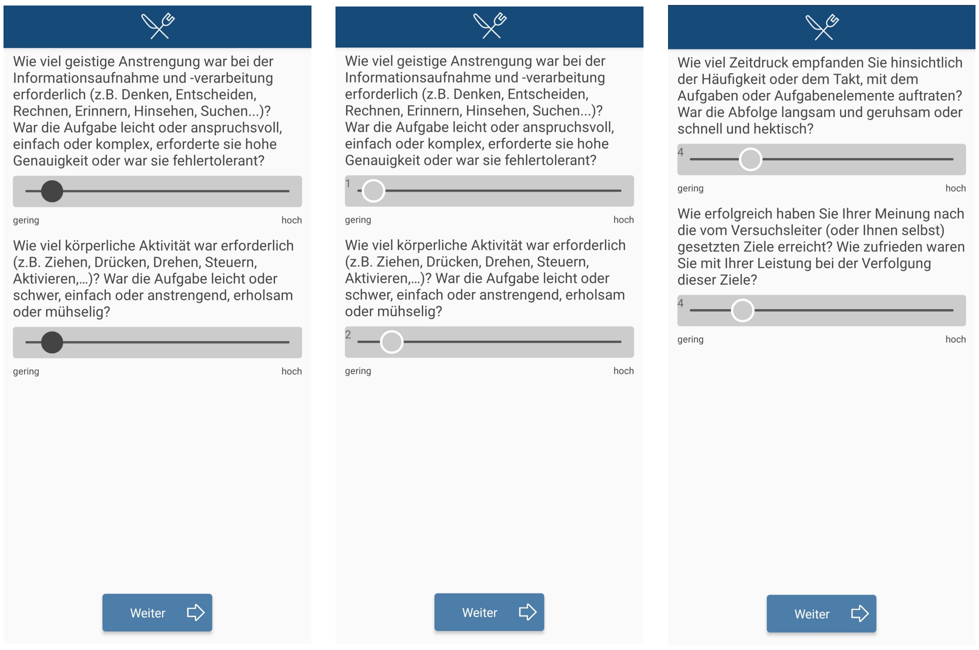
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.

[✓ ●] ● ● ● ●

Stimme überhaupt nicht zu — Stimme voll und ganz zu

Weiter

Figure B.5: System Usability Scale Questionnaire in the mobile application


Wie viel geistige Anstrengung war bei der Informationsaufnahme und -verarbeitung erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen...)? War die Aufgabe leicht oder anspruchsvoll, einfach oder komplex, erforderte sie hohe Genauigkeit oder war sie fehlertolerant?

gering — hoch

Wie viel körperliche Aktivität war erforderlich (z.B. Ziehen, Drücken, Drehen, Steuern, Aktivieren,...)? War die Aufgabe leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

gering — hoch

Weiter

Wie viel geistige Anstrengung war bei der Informationsaufnahme und -verarbeitung erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen...)? War die Aufgabe leicht oder anspruchsvoll, einfach oder komplex, erforderte sie hohe Genauigkeit oder war sie fehlertolerant?

1

gering — hoch

Wie viel körperliche Aktivität war erforderlich (z.B. Ziehen, Drücken, Drehen, Steuern, Aktivieren,...)? War die Aufgabe leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

2

gering — hoch

Weiter

Wie viel Zeitdruck empfanden Sie hinsichtlich der Häufigkeit oder dem Takt, mit dem Aufgaben oder Aufgabenelemente auftraten? War die Abfolge langsam und geruhsam oder schnell und hektisch?

4

gering — hoch

Wie erfolgreich haben Sie Ihrer Meinung nach die vom Versuchsleiter (oder Ihnen selbst) gesetzten Ziele erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

4

gering — hoch

Weiter

Figure B.6: NASA-TLX Questionnaire in the mobile application

C ADDITIONAL GRAPHICS

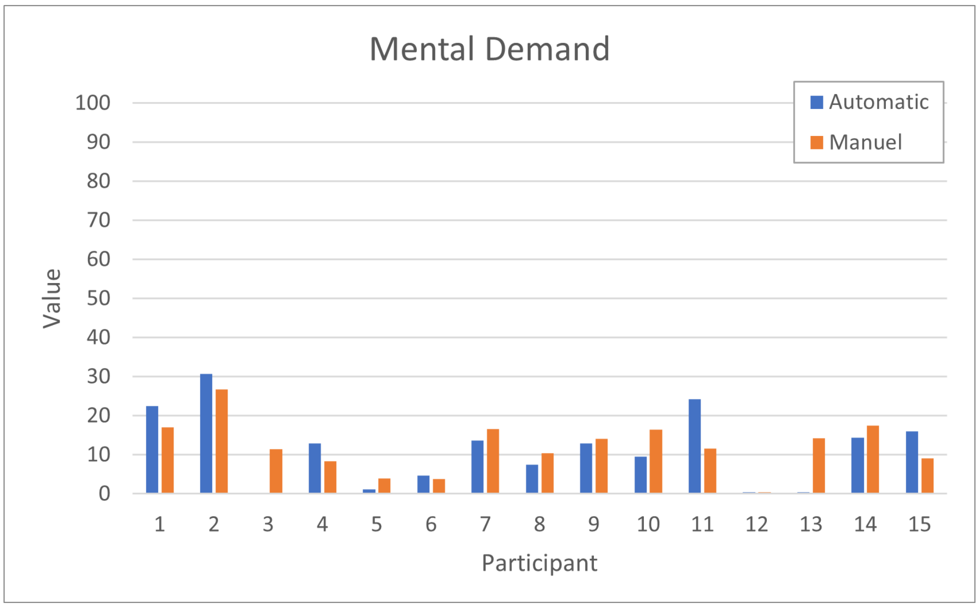


Figure C.1: Results of the NASA-TLX questionnaire for mental demand

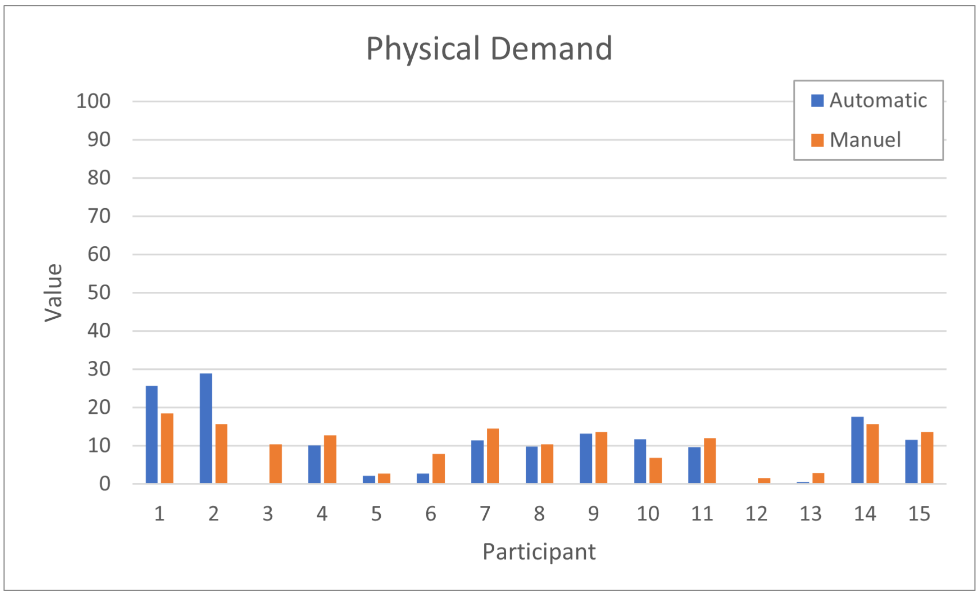


Figure C.2: Results of the NASA-TLX questionnaire for physical demand

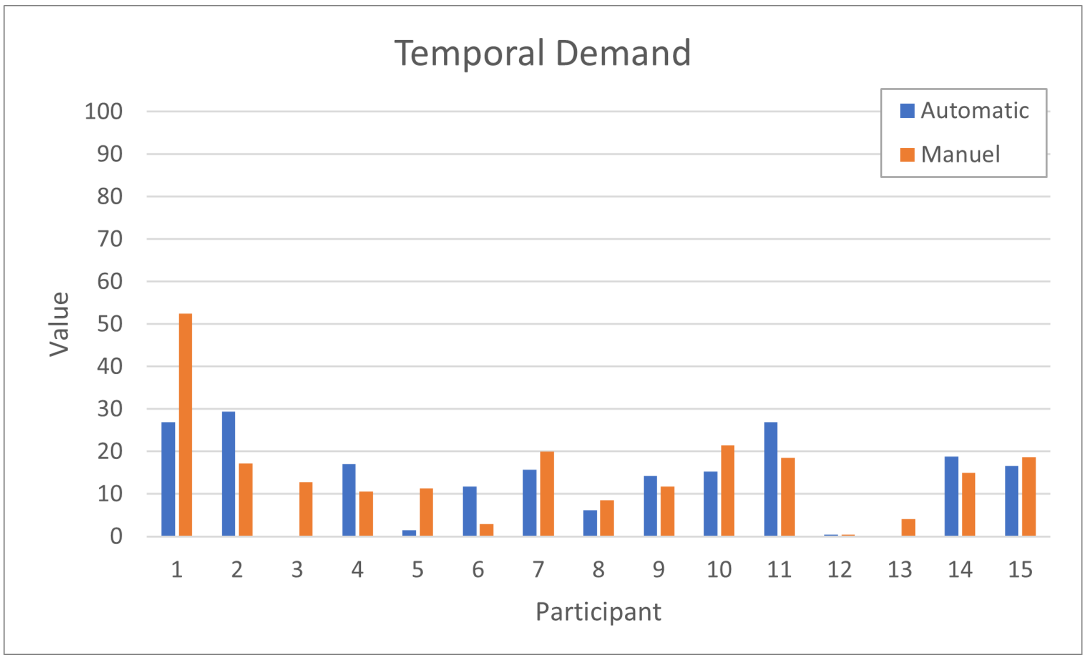
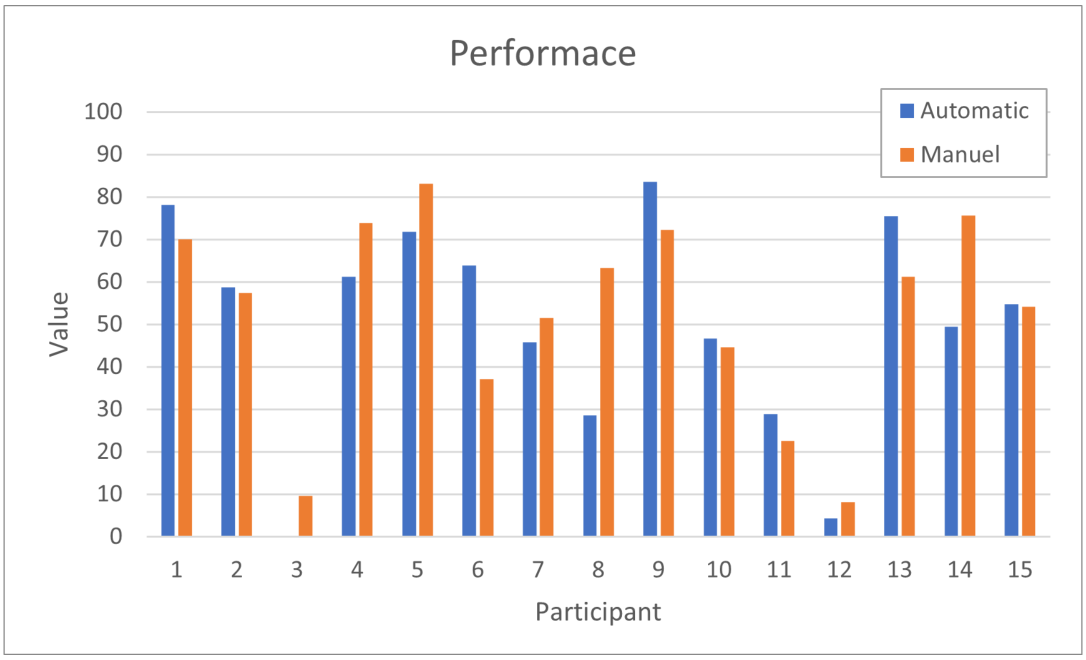


Figure C.3: Results of the NASA-TLX questionnaire for temporal demand



Results of the NASA-TLX questionnaire for performance

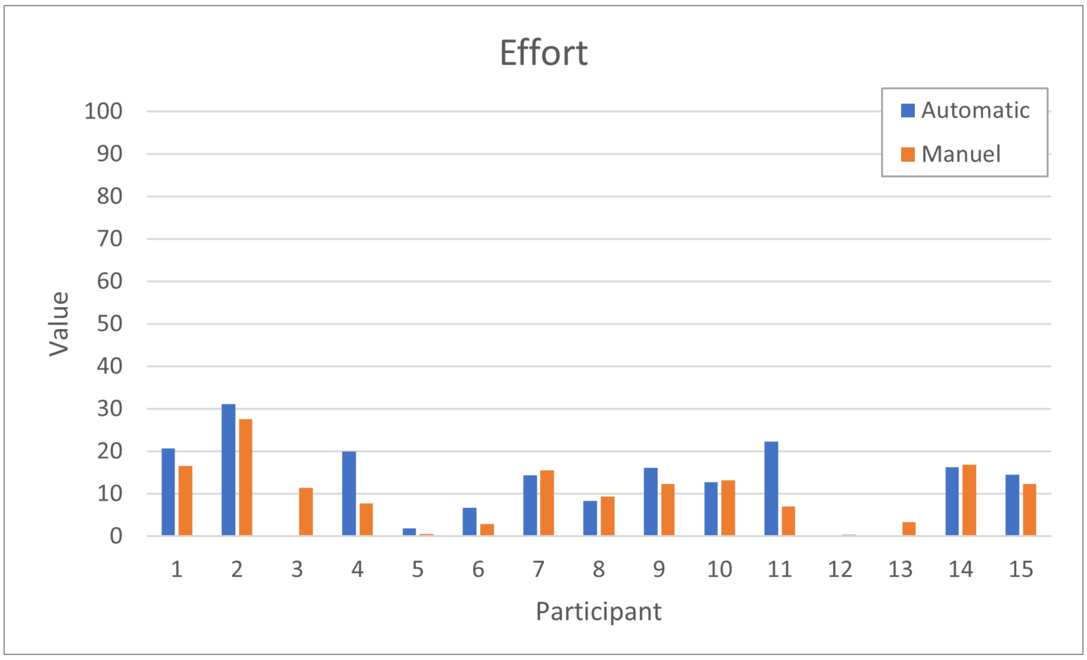


Figure C.4: Results of the NASA-TLX questionnaire for effort

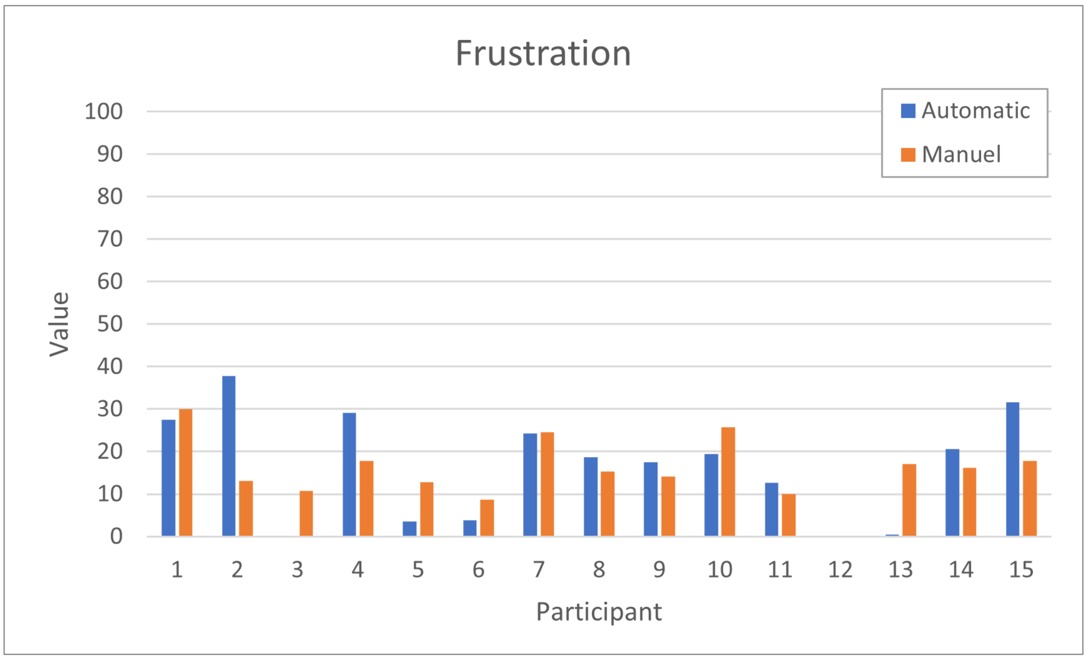


Figure C.5: Results of the NASA-TLX questionnaire for frustration

D ENCLOSURE

The attached DVD includes all relevant files for this work:

- Files to perform the mean shift algorithm used in this work to perform the pre-segmentation
- Digital version of this document
- Files for the custom MTurk job
- Docker and Kubernetes files to run the Mask R-CNN model training
- The apk for the smartphone application
- Two KNIME workflows:

  - AnalyseMturkWithClasses.knwf - A workflow to review the results of the MTurk job

  - AnalyseDBs.knwf - A workflow, which is used to get the results of the usability study