Master Thesis

September 11, 2020

# Retail Product Classifier

## A mobile app for retail packaged product image classification and dataset management

**Marios Visos**

of Volos, Greece (17-717-927)

**supervised by**

Prof. Dr. Harald C. Gall

Dr. Pasquale Salza

University of Zurich[UZH]

ETH

Eidgenössische Technische Hochschule Zürich

Swiss Federal Institute of Technology Zurich

s.e.a.l.

software evolution & architecture lab

Master Thesis

# Retail Product Classifier

## A mobile app for retail packaged product image classification and dataset management

**Marios Visos**

University of Zurich[UZH]

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

s.e.a.l.
software evolution & architecture lab

**Master Thesis**

**Author:** Marios Visos, marios.visos@uzh.ch

**Project period:** 11.03.2020 - 11.09.2020

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

# Acknowledgements

First and foremost, I would like to thank Klaus Fuchs for his inspiration, fruitful ideas and great support throughout my thesis as well as Prof. Elgar Fleisch for being my supervisor. I would also like to thank Prof. Harald Gall for giving me the opportunity to write this master thesis at the software evolution and architecture lab (s.e.a.l.) at University of Zurich in collaboration with Prof. Elgar Fleisch and the Information Management research group of the Department of Management, Technology, and Economics at ETH Zurich. Many thanks to Dr. Pasquale Salza for his helpful hints, input and guidance for this project. Moreover, I would like to thank my girlfriend for assisting me throughout this research work as well as my sister for motivating me especially towards the end of this thesis. Last but not least, I would like to thank my parents for supporting me not only throughout my academic career but also through my whole life.

# Abstract

Research in computer vision on packaged products is often facing the problem of lacking publicly available labelled datasets. Deep Convolutional Neural Network's performance heavily depends on the quality as well as the quantity of the data used for training. Indeed, many datasets are struggling with issues such as becoming quickly outdated, being collected in conditions not matching the real-world environment or being small in size. This thesis is focused on the design and implementation of a software system to ease the traditional, tedious process of manually generating and managing meaningful labelled datasets. The software system includes a mobile client, an application server, and two computer vision components used for object detection. The mobile application gives users the power to generate and manage labelled image datasets, including useful metadata, and persist them in a database via the application server. In addition, users can easily trigger actions such as training, validating, testing as well as performing predictions on new unseen images. These actions are available in the mobile application and can be triggered with the help of the computer vision components. The computer vision components are integrated into the software system in a way that allows them to be easily replaced with different deep Convolutional Neural Network architectures. To evaluate the quality of the devised software system, we conducted a usability study in collaboration with a Swiss company named Valora at one of their supermarket stores called "avec" located at the Zurich main station. The overall results of our usability study were positive, and the feedback acquired from our 14 participants' answers to our questionnaire render our mobile application valuable. As a result of our study, a publicly available dataset was generated, including a total of 88 product labels and 2630 images. Finally, we evaluated the generated datasets by using them as training data in two models, one for object localization and another for object detection. The results of testing the object detection model with our datasets make our application a sufficient replacement for the manual process of creating and annotating datasets used in an image classification task.

# Zusammenfassung

Die Forschung auf dem Gebiet der Computervision an verpackten Produkten steht oft vor dem Problem, dass öffentlich zugängliche, gekennzeichnete Datensätze fehlen. Die Leistung von Deep Convolutional Neural Network hängt sowohl von der Qualität als auch von der Quantität der für das Training verwendeten Daten ab. Viele Datensätze haben in der Tat mit Problemen zu kämpfen, wie z.B. der schnellen Veralterung, der Erfassung unter Bedingungen, die nicht der realen Umgebung entsprechen, oder der geringen Grösse. Diese Arbeit konzentriert sich auf den Entwurf und die Implementierung eines Softwaresystems, das den traditionellen, mühsamen Prozess der manuellen Generierung und Verwaltung von aussagekräftigen, gekennzeichneten Datensätzen erleichtert. Das Softwaresystem umfasst einen mobile-Anwendungs-Client, einen Anwendungsserver und zwei Bildverarbeitungskomponenten zur Objekterkennung. Die mobile Anwendung gibt den Benutzern die Möglichkeit, etikettierte Bilddatensätze, einschliesslich nützlicher Metadaten, zu erzeugen und zu verwalten und sie über den Anwendungsserver in einer Datenbank zu speichern. Darüber hinaus können Benutzer auf einfache Weise Aktionen wie Training, Validierung, Testen sowie die Durchführung von Vorhersagen für neue, ungesehene Bilder auslösen. Diese Aktionen stehen in der mobilen Anwendung zur Verfügung und können mit Hilfe der Computer Vision-Komponenten ausgelöst werden. Die Bildverarbeitungskomponenten sind so in das Softwaresystem integriert, dass sie leicht durch verschiedene tiefe Convolutional Neural Network-Architekturen ersetzt werden können. Um die Qualität des entwickelten Softwaresystems zu evaluieren, haben wir in Zusammenarbeit mit einer Schweizer Firma namens Valora eine Usability-Studie in einem ihrer Supermärkte namens "avec" am Hauptbahnhof Zürich durchgeführt. Die Gesamtergebnisse unserer Usability-Studie waren positiv, und das Feedback, das wir durch die Antworten unserer 14 Teilnehmer auf unseren Fragebogen erhielten, macht unsere mobile Anwendung wertvoll. Als Ergebnis unserer Studie wurde ein öffentlich zugänglicher Datensatz generiert, der insgesamt 88 Produktetiketten und 2630 Bilder umfasst. Schliesslich werteten wir die generierten Datensätze aus, indem wir sie als Trainingsdaten in zwei Modellen verwendeten, eines für die Objektlokalisierung und eines für die Objekterkennung. Die Ergebnisse des Testens des Objekterkennungsmodells mit unseren Datensätzen machen unsere Anwendung zu einem ausreichenden Ersatz für den manuellen Prozess der Erstellung und Annotation von Datensätzen, die bei einer Bildklassifikationsaufgabe verwendet werden.

# Contents


**1 Introduction** **1**

**2 Related Work** **5**
2.1 Image classification . . . . . 5
2.2 Object detection . . . . . 7

**3 Software system design** **11**
3.1 Interactive prototype . . . . . 11
3.2 System overview . . . . . 13
3.2.1 Mobile application . . . . . 13
3.2.2 Computer vision . . . . . 14
3.2.3 Application server . . . . . 14
3.2.4 External APIs . . . . . 15

**4 Implementation** **17**
4.1 Front-end . . . . . 17
4.1.1 Sign In Screen . . . . . 17
4.1.2 Home Screen . . . . . 18
4.1.3 Dataset Screen . . . . . 19
4.1.4 Label Screen . . . . . 21
4.1.5 Image Screen . . . . . 22
4.1.6 Camera Screen . . . . . 24
4.2 Back-end . . . . . 28
4.2.1 Application server . . . . . 28
4.2.2 Computer vision . . . . . 31

**5 Evaluation** **41**
5.1 Study settings . . . . . 41
5.2 Results . . . . . 44
5.2.1 Dataset . . . . . 44
5.2.2 Usability study . . . . . 44
5.2.3 Dataset evaluation . . . . . 45

**6 Discussion** **55**

**7 Conclusion** **59**

**Appendix** **65**

# List of Figures

2.1 System diagram of object detection proposed by Goldmann et al. [1] . . . . . 7
2.2 Visualization of the output of the EM-Merger unit from Goldmann et al. [1] . . . . 10

3.1 Mobile app mockup on the whiteboard . . . . . 12
3.2 Interactive wireframe of the app built with Balsamiq . . . . . 13
3.3 Entity relationship model diagram . . . . . 15
3.4 System overview . . . . . 16

4.1 Sign In Screen . . . . . 17
4.2 Home Screen . . . . . 19
4.3 Dataset Screen . . . . . 20
4.4 Label Screen . . . . . 21
4.5 Image Screen . . . . . 22
4.6 Camera Screen . . . . . 24
4.7 NearbyStores Component . . . . . 25
4.8 BarCodeScan component . . . . . 26
4.9 BoundingBox component . . . . . 27
4.10 The Triplet Loss tries to minimize the distance between an anchor and a positive, both of which are from the same product, and maximizes the distance between the anchor and a negative of a different product [2] . . . . . 37

5.1 Questionnaire first page . . . . . 42
5.2 Questionnaire second page . . . . . 43
5.3 The results of the first question . . . . . 44
5.4 The results of the second question . . . . . 45
5.5 The results of the third question . . . . . 46
5.6 The results of the fourth question . . . . . 47
5.7 The results of the fifth question . . . . . 48
5.8 The results of the sixth question . . . . . 49
5.9 Result image of testing the object localization model provided by Goldmann et al. [1] without training it with any of our data . . . . . 50
5.10 Result image of testing our object localization model trained with our data using transfer learning from Goldmann et al.'s [1] model . . . . . 51
5.11 Result image of testing our model trained with our data using transfer learning from Lin et al.'s [3] model . . . . . 52
5.12 Histogram of the classification accuracy of 460 predicted bounding box instances of Lin et al.'s [3] model trained on our manually annotated dataset . . . . . 53

6.1 The Mean Average Precision (mAP) of five detectors training under different instance-level missing label rates of the training dataset. The amount of missing annotations has a significant impact on the performance of the detectors [4]. . . . . 57

# List of Tables

4.1 Table containing the CSV data format for the annotation file used for training the object localization model . . . . . 33
4.2 Table containing the CSV data format for the annotation file used for the object detection model . . . . . 40

4.3 Table containing the CSV data format of the inference output file of the object detection model . . . . . 40

# List of Listings

4.1 Utility function for converting our database annotations into a CSV needed for the object localization training algorithm. . . . . 33
4.2 Utility function for splitting the annotations into train validation . . . . . 35
4.3 Utility function for generating triplets. . . . . 37

# Chapter 1

# Introduction

Identification of packaged products via computer vision is really promising according to current studies [5] [6] [7] [8]. A major constraint faced by the research in computer vision on packaged products is the absence of publicly available labelled datasets since there are owners who wouldn't make their image data public [9]. As most deep learning problems, it strongly relies on the quality of the training data that you feed in the deep learning models. However, quite some of the existing datasets become quickly out of date, not big in size [10] [6], or were collected under lab conditions that do not always match the real-world environment [9] [11]. Nevertheless, the absence of publicly free datasets of images makes the comparison across different packaged product identification methods more challenging [12].

Since a huge amount of training data is needed to learn complex invariances, training a new CNN (Convolutional Neural Network) from scratch might not be enough for having favorable results. Fortunately, several object detection problems have tackled this challenge by using the so-called transfer learning approach, in which an already developed model is reused as the starting point for another model. Their work has shown that by using transfer learning we can have CNN intermediate features which are general enough so that they can be applied to different problems [11] [13]. Although the improvement of the performance of deep learning algorithms shows remarkable progress [14] [15], improvements in dataset construction are inadequate and the already available datasets are lagging behind. The deep learning models are getting deeper and the number of parameters in many of them is bigger than the number of images in these datasets [16]. Although models are deepening [17] [15] and computation power is growing, the increase of the dataset size for training and evaluating is left behind instead of being in line with the models and computation power growth. For quite some applications in computer vision, a high-quality training image dataset is assumed [18]. Better image quality does not always lead to a better dataset but what is more important, is that the images collected for training are as close as possible to the real-world environment.

In order to tackle the above problems in the field, the goal of this study is to facilitate the tedious process of collection and management of packaged product datasets in a retail environment. On the one hand, this thesis introduces a cross-platform mobile application to be used as a retail packaged product image classifier and dataset manager. The technical feasibility of such a software system will be examined and assessed by exploring what makes a good quality dataset, and how we can guide the user to create a valuable dataset not only for his own project but one which is meaningful for similar projects as well. On the other hand, there will be a user study in order to test the usability of the app and have the ability to evaluate and compare the quality of data generated from different users.

Based on the goal of this thesis, this study constitutes the following research questions:

**RQ1: Can a mobile application reinforce a faster and more meaningful generation of labelled Convolutional Neural Network datasets?**

**RQ2: Is a mobile application sufficient for replacing the manual creation and annotation of datasets for both object localization and object classification while having similar accuracy?**

To answer these research questions, first, literature analysis of existing work in this area was performed. Second, we developed a software system that consists of a mobile application client for retail packaged product image classification and dataset management backed by an application server. To assess the quality of our software system, a usability study was performed with 14 participants at a supermarket store called "avec", located at the Zurich main station which is part of a Swiss company named Valora. The users were asked to fill in a questionnaire to acquire valuable feedback from them regarding our application. The datasets collected from the users were fed into two different deep neural network architectures in order to assess their quality. Our usability study had positive results and the questionnaire answers showed that the users overall enjoyed using our mobile application and found it useful. A dataset of four different supermarket shelves was generated including 88 product labels and 2630 images with 3257 bounding boxes.

The following are the major contributions of this thesis:

- A cross-platform mobile application used for collecting and organizing datasets, training deep neural network models with them as well as making predictions on new unseen data using their already trained models for inference.
- A usability study was conducted under real supermarket store conditions with positive overall feedback from the users regarding their experience with the developed mobile application.
- A dataset of 2630 images of supermarket shelves containing 88 different product labels and 3257 bounding box annotations.
- Results of testing the object detection model by Lin, Goyal, Girshick, He & Dollár et al. [3] using our generated datasets were positive rendering our software system a good replacement for the image classification task.

The rest of the thesis is structured as follows:

- Chapter 2 provides a review of research work closely coupled with this thesis. The related work is mainly in the areas of image classification along with object detection in the retail domain and specifically on the identification of packaged goods.
- Chapter 3 analyses the research design of the major components of the developed software system and indicates how they are connected. The whole lifecycle of the design is presented starting from a whiteboard based prototype, moving to an interactive prototype, and finally describing the system overview of the major components.
- Chapter 4 presents our methods for tackling the research questions mentioned above. We begin with the implementation details of our front-end part being the mobile client application. We continue with an overview of our back-end which consists of our application server as well as the computer vision components.
- Chapter 5 describes our two-folded evaluation of our research. First, the study settings followed by the results of the study are presented. In addition, we perform an evaluation of the datasets generated by the usability study by highlighting the results of the computer vision components trained with those datasets.

- The thesis is concluded in chapter 6 with a discussion regarding our results, the limitations of our work followed by possible future improvements.

# Chapter 2

# Related Work

## 2.1 Image classification

Image classification is a subset of computer vision that aims at classifying images according to their content. In our case, given an image of a product, the image classification would return the exact name of that product. One of the first papers to study image classification, specifically in the retail domain, was by Belongie, Merler& Galleguillos et al. [19]. The authors' main contributions include the release of a multimedia dataset, named GroZi-120, and a thorough analysis of their object recognition algorithms proposed for applications in either mobile robot navigation or as assistance for the visually impaired. Using the GroZi-120 dataset in the work of Winlock, Christiansen& Belongie et al. [20], the authors suggest an approach that would allow the visually impaired shop at a supermarket store without help from another human. They use a multi-class naive-Bayes classifier inspired by NIMBLE [21], which is trained on enhanced SURF [22] descriptors extracted from images in the GroZi-120 dataset. Afterward, it is used to compute per-class probability distributions on video key points for final classification. While most papers focus on a relatively small-scale problem [23], the approach for multi-label image classification presented in the work of George& Floerkemeier et al. [24] is more pragmatic. In their cross-dataset classification, they used only one image per product in their training set. Besides, they proposed a dataset that they made publicly available comprising of 8350 images of products including 80 labels and 680 high-quality shelf images.

Tonioni et al.'s [23] work, which specialized in image classification on the identification of packaged products on supermarket shelves and used a domain-invariant embeddings approach, has been an inspiration for our work. This approach assumes that the bounding boxes containing the products are detected at an earlier stage and then the content of the boxes is cropped. Therefore, from one image with multiple objects, we generate one new image per object by cropping the content of its bounding box from an object localization result. The individual shelf image is called a query image and an e-commerce image with a white background is called a reference image. Tonioni et al.'s [7] pipeline tries to find which corresponding reference image has the most similar appearance to the query image of a product.

On a high level, in order to learn a global embedding vector for every image, a VGG-16 CNN is used [8]. When predicting new unseen images of products, the trained model is used to calculate an embedding vector for the query image. By using the nearest method search the nearest reference embedding vector to the query image is found. Then, a so-called image-to-image translation GAN is used so that instead of training the VGG-16 with studio-quality images, to be used for training it with in-situ images, which most of the time have different lighting, scale, viewpoint, occlusion when inferring. Particularly, the product appearance is not changed much by the GAN

while it generates an in-situ-like version of each image. By using a triplet ranking loss while training, the VGG-16 is trying to minimize the distance between a GAN-generated image's embedding in the embedding vector space and its reference image's embedding. At the same time, its goal is also to maximize the distance between those two and reference images of different products making the VGG-16 particularly robust to the domain-shift [25].

On a low level, the encoder can be viewed as a function that takes an image and returns a k-dimensional embedding vector, which is also called a descriptor. In addition to sample generation similar to the test domain, the image translation GAN tries to generate hard instances in order to make the encoder learning more difficult. The following triplet loss function is then used for the encoder in order to be trained:

$$\mathcal{L}_{enc} = max(0, d(E(i_\alpha), E(i_p)) - d(E(i_\alpha), E(i_n) + \alpha) \tag{2.1}$$

Where $E : \mathcal{I} \mapsto \mathcal{D}$ maps an image $i \in \mathcal{I}$ to descriptor $d^k \in \mathcal{D}$ and $d$ is used as a metric for distance in the image descriptor space and is defined as: $d(x, y) = 1 - xy$ for $x, y \in \mathcal{D}$, a is a fixed margin. The $i_a$ sample is an in-situ simulation generated by its GAN, $i_p$ refers to a reference image, and $i_n$ is a sample from a different class. In the 2.1 equation, we need triplets of samples for the loss function. One image is generated using the generator $i_a^B = \mathcal{G}(i_p^A)$ which is a mapping $\mathcal{G} : \mathcal{A} \mapsto \mathcal{B}$. We then have two images from the same domain $i_p^A$, $i_a^A \in \mathcal{A}$. $\mathcal{A}$, which is the domain for e-commerce quality reference images, and $\mathcal{B}$, which stands for the domain of the in-situ images are both subsets of $\mathcal{I}$. As a result, the 2.1 equation includes two images from domain $\mathcal{A}$ and one generated image from the $\mathcal{B}$. The discriminator D, which is defined as a mapping $\mathcal{D} : \mathcal{I} \mapsto \mathbb{R}$, is aiming for minimizing the following cross-entropy loss:

$$\mathcal{L}_{disc} = \log(\mathcal{D}(i^B)) + \log(1 - \mathcal{D}(\mathbb{G}(i_p^A))) \tag{2.2}$$

The generator on the other hand minimizes the following more compound loss:

$$\mathcal{L}_{gen} = L_{adv} + \lambda_{reg} L_{reg} + \lambda_{emb} L_{emb} \tag{2.3}$$

Where:

$$L_{adv} = -\log(\mathcal{D}(\mathbb{G}(i_p^A))) \tag{2.4}$$

$$L_{reg} = \phi(i_p^A, i_a^B) \tag{2.5}$$

$$L_{emb} = -d(E(i_p^A), E(G(i_p^A))) \tag{2.6}$$

$\phi(x, y), x, y \in \mathcal{I}$ is defined as the similarity measure $\phi(x, y) = ZNCC(x, y)$ (Zero Mean Normalized Cross-Correlation). We also have two hyperparameters $\lambda_{reg}$ and $\lambda_{emb}$ which control the effect of their associated terms. $\mathcal{L}_{emb}$ stimulates the desired GAN's behaviour to generate samples that would be difficult for the encoder network to learn. $\mathcal{L}_{adv}$ is a standard adversarial loss and $\mathcal{L}_{reg}$'s goal is to encourage the GAN to generate similar images to the input images so that collapsing, which is a common problem in GANs, can be prevented [25].

## 2.2 Object detection

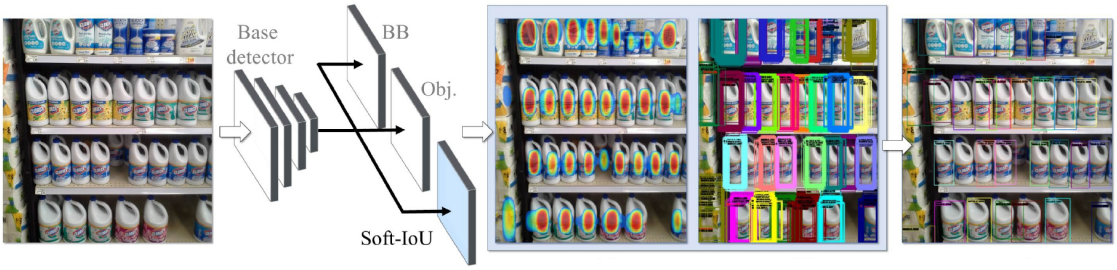



**Figure 2.1**: System diagram of object detection proposed by Goldmann et al. [1]

One of the latest state of the art object detection architecture in densely packed scenes and specifically in the retail domain was introduced by Goldmann et al. [1]. They proposed adding two novel algorithmic components named heads to the detector as seen in figure 2.2. The first component is called the regression head and learns to detect potential regions of objects by extracting various bounding box coordinates. The classification head learns to differentiate between regions that look like positive objects and ones which look like negative objects or negative backgrounds. Therefore, in this binary simple case, we learn both examples which contain objects and backgrounds without the object. On top of these components, the authors proposed adding another metric for estimating the quality of each detection by computing the overlapping rate or intersection over union (IoU) (see 2.7) for each detection which tells us how much the detection overlaps with the actual box. Since the IoU value is between 0 and 1 they give a probabilistic interpretation of this IoU (see 2.8) rate function and this is learned with the binary cross-entropy loss (see 2.8). Consequently, two different scores are learned for each detection and are used to evaluate the quality of each detection. One score tells us how much a detection looks like an object and the other score from the IoU head estimates how much the detection overlaps with the nearby objects. By combining these two different ways to measure the quality of the localization from different aspects we get more information and knowledge for estimating the quality of each detection.

The IoU between a predicted box $b_i$ and a ground truth box $\hat{b_i}$ where $N$ is the number of predicted detections and $i \in \{1..N\}$ is defined as:

$$IoU_i = \frac{Intersection(\hat{b_i}, b_i)}{Union(b_i, \hat{b_i})} \tag{2.7}$$

Probabilistic interpretation, learning it with the Soft-IoU layer using a binary cross-entropy loss:

$$\mathcal{L}_{sIoU} = -\frac{1}{n}\sum_{n=1}^{n}[IoU_i \log(c_i^{iou}) + (1 - IoU_i)\log(1 - c_i^{iou})] \tag{2.8}$$

where n is the number of samples in each batch.

In the detection network the following loss is used to train each RPN:

$$\mathcal{L} = \mathcal{L}_{\text{Classification}} + \mathcal{L}_{\text{Regression}} + \mathcal{L}_{\text{sIoU}} \tag{2.9}$$

Where, $\mathcal{L}_{\text{Classification}}$ and $L_{\text{Regression}}$ are the standard cross-entropy and euclidean losses, respectively [26] [27] [28], and LsIoU is defined in 2.8.

The network of 2D Gaussians is defined as the following set:

$$F = \{f_i\}_{i=1}^{N} = \{\mathcal{N}(p; \mu_i, \Sigma_i)\}_{i=1}^{N} \tag{2.10}$$

where N is the number of bounding boxes produces by the network and $p \in \mathbb{R}^2$ a 2D image coordinate.

The central point of the box, a 2D mean, the central point of the box $\mu_i = (x_i, y_i)$ and a diagonal covariance $\Sigma_i = [(h_i/4)^2, 0; , 0, (w_i/4)^2]$ alltogeter represent the $i$-th detection. These Gaussians are then presented jointly, as a single MoG density:

$$f(p) = \sum_{i=1}^{N} a_i f_{i(p)} \tag{2.11}$$

where:

$$\alpha_i = \frac{c_i^{\text{iou}}}{\Sigma_{k=1}^{N} c_k^{\text{iou}}} \tag{2.12}$$

represent the mixture coefficients which depict the confidence that the bounding box overlaps with its ground truth, and are normalized to create a MoG.

The problem of resolving the final detections is converted in finding a set of $K << N$ Gaussians:

$$G = \{g_j\}_{j=1}^{K} = \{\mathcal{N}(p; \mu'_j|, \Sigma'_j)\}_{j=1}^{K} \tag{2.13}$$

so that the selected Gaussians approximate the original MoG distribution f of 2.11, formed by all N detections, when aggregated.

Therefore, if g is defined as:

$$g(p) = \sum_{j=1}^{K} \beta_j g_j(p) \tag{2.14}$$

we then try to find a mixture of K Gaussians, G, for which

$$d(f, g) = \sum_{i=1}^{N} \alpha_i \min_{j=1}^{K} KL(f_i||g_j) \tag{2.15}$$

is minimized where where KL is the KL-divergence used as a non-symmetric distance between two detection boxes.

The E-step assigns each box to the closest box cluster, where a KL distance defines the box similarity between the corresponding Gaussians. E-step assignments are defined as:

$$\pi(i) = \arg\min_{j=1}^{K} KL(f_i||g_j) \tag{2.16}$$

The model parameters are then re-estimated by the M-step:

$$\beta_j = \sum_{i \in \pi^-1(j)} \alpha_i \tag{2.17}$$

$$\mu'_j = \frac{1}{\beta_j} \sum_{i \in \pi^-1(j)} \alpha_i \mu_i \tag{2.18}$$

$$\Sigma'_j = \frac{1}{\beta_j} \sum_{i \in \pi^{-1}(j)} \alpha_i (\Sigma_i + (\mu_i - \mu'_j)(\mu_i - \mu'_j)^T). \tag{2.19}$$

The second component they implemented was the EM-Merger whose goal was to remove the overlapping duplicate bounding boxes around an object. Normally Non-Maximum Suppression (NMS) is used but in the case of densely packed scenes, it does not solve well the overlapping issue. Therefore, the authors proposed their own algorithm for solving the overlapping bounding boxes problem. First, they convert each box into a 2d Gaussian where the center of the box corresponds to the center of the Gaussian and the dimensions of the box correlate to the dimensions of the Gaussian. In addition, the score every box gets from the neural network is correlated to a weight assigned for each one of the Gaussians. This results in a linear combination of a set of all the weighted Gaussians and after normalization, we get a probabilistic distribution of a mixture of Gaussians. Afterward, expectation-maximization is applied in order to convert this mixture of many overlapping Gaussians into a mixture of few distinct Gaussians which are later translated into the final bounding box results. This can be better understood in figure 2.2 where we see the mixture of Gaussians which is visualized as a heat-map. This heat-map in the background of objects is an aggregation of hundreds of overlapping Gaussians and the green ellipsoids visualize the result which is a few distinct Gaussians. These actually resolve all the duplicates as each ellipsoid is localized around individual objects and by converting these ellipsoids back into boxes we have our final results.

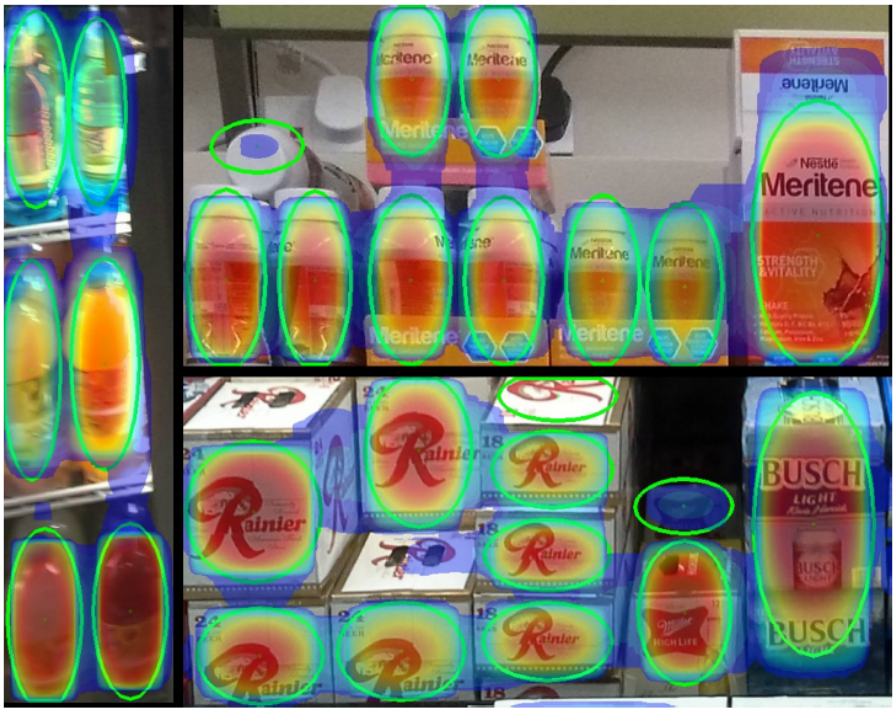



**Figure 2.2**: Visualization of the output of the EM-Merger unit from Goldmann et al. [1]

# Chapter 3

# Software system design

In this chapter, I present the research design of the major components of my software system as well as their relation. The main purpose of my application is to enforce the meaningfulness and re-usability of the generated datasets for other computer vision problems focused on the identification of packaged products. All the features of my system's components are developed in a way so that they can add their own value in fulfilling that purpose.

## 3.1 Interactive prototype

On the first stage, before I even started coding, I wanted to design and develop an interactive prototype. The goal of having a prototype is that it can help us have a better understanding of the basic concepts of the app. Furthermore, it aids not only in identifying whether all the requirements are met but also in highlighting what should be corrected. The whole process of designing the prototype started on a whiteboard. Although the first iteration of the mockup screens was quite basic, it served as a good foundation for building the app at later stages. As you can see in figure 3.1, initially we had lists of datasets and labels. Since the application is focused on the retail domain, it made sense to rename datasets into shelves so that the users can focus on completing datasets related to specific shelves first. After the whiteboard prototype, a mockup software solution was needed to better visualize the app's functionalities and ease the process of getting feedback from others.

There are a plethora of prototyping tools out there but for my app, I decided to use a software called Balsamiq for several reasons. First, it is easy to wireframe since the user can just drag-and-drop elements and combine them in order to create a screen. These User Interface (UI) elements are pre-designed and ready to be used so one does not have to come up with the design from scratch. Alternatively, just by using a drag and drop option, someone can conveniently create screens. Furthermore, since my focus for this mobile app was not so much on the graphic design but its functionality, Balsamiq, being a low-fidelity prototyping tool, perfectly fitted my needs. Another great feature of my wireframe is that you can make it interactive by linking the screens together and have clickable elements, e.g. buttons, which after they are pressed, they navigate to a new screen. After some raw iterations both on blank paper and whiteboard first, I implemented a first version of the wireframe in Balsamiq. I then shared this first draft with some users who could interactively navigate through different screens and write comments in different sections stating what they like and dislike about the prototype. Consequently, at an already early stage before the development of the app, together with some iterative testing and feedback from users, I discovered mistakes and improvements in the app's design which saved me quite some time

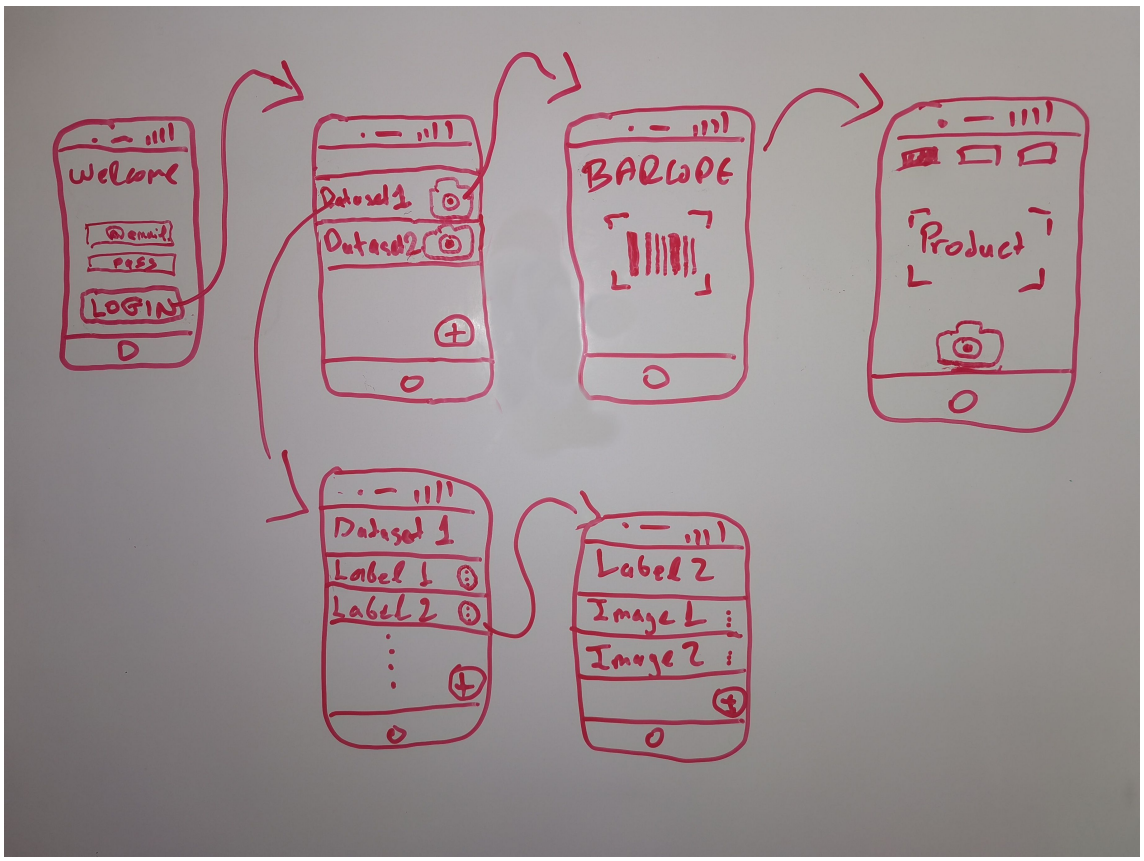



**Figure 3.1**: Mobile app mockup on the whiteboard

during development. Finally, after iterating over and considering the valuable feedback from users, the final version of the app's prototype can be seen in figure 3.2.


**Figure 3.2**: Interactive wireframe of the app built with Balsamiq

## 3.2 System overview

In this section, I provide a high-level overview of my software system's major components, that can be seen in figure 3.4, and how they are connected. Besides, I present the technology stack by describing which technologies are used and why they were chosen for every component of our software system.

### 3.2.1 Mobile application

The mobile application serves as a client for generating and managing datasets ready to be fed in both localization and classification deep neural networks. It is a tool for users with which they can organize, view, and edit all their dataset labels and images. Furthermore, by using tutorial-

like hints, it guides the user step by step to make the most out of the app and generate datasets enriched with useful metadata. Nowadays, there is a wide range of libraries and frameworks for developing mobile applications. On a high level, I had to choose between three options, native libraries for Android (Java/Kotlin) or iOS(Objective-C/Swift), a cross-platform solution like React Native which compiles to native code, or a framework used for building web apps which are then wrapped by a real native app that hosts a webview. From these three options, the native code development is the most performant. However, it limits you since your app will run on either Android or iOS and you will need to completely re-develop the whole app if you need to target the other platform. The "wrapped web app" runs on both Android and iOS operating systems but lacks a bit on performance since it is a web app running on a phone. Two major requirements of my app were the quick generation of a lot of training data and the app's performance since the user makes heavy use of the phone's camera. React Native, which is a framework based on a JavaScript library called React developed by Facebook, is a great balance for my trade-off between the fast generation of massive amounts of data and good performance. One the one hand, the fact that it is cross-platform allows users regardless of their phone's operating system to use the app which improves the speed of data collection. On the other hand, since it is not a wrapped web app but rather offers a real native mobile app compiled from your code, its performance is one of the best you can get for a cross-platform mobile application.

### 3.2.2 Computer vision

Two main components of my software system are product detection and classification. My focus has been on the rest of my software system which aims to be a tool for people who would like to work on building state of the art deep neural network models on the identification of packaged goods. Therefore, the CNNs that I use are included in a modular way in my system meaning that they can easily be replaced with pipelines other than my selected ones. Basic functionalities of these CNNs like training, validating, testing as well as classifying new unseen products are communicated by using my back-end server as a RESTful API, one that conforms to the representational state transfer (REST) software architectural style. As I already mentioned, there are a plethora of technologies one can use for both image classification and object localization. My concentration is on the retail domain and the latest state of the art detection of densely packed objects in this field was implemented by Goldmann et al. [1]. Their implementation was used just for the object detection part. Then inspired by Mukhija et al. [25] I crop the images by using the detected bounding boxes and use the approach from Hoffer& Ailon et al. [29] for performing image classification for each detected product. Since the work of Goldmann et al. [1] is built on top of RetinaNet object detection as described by Lin et al. [3] which includes both object localization and classification at the same time this was also separately included and tested in my software system.

### 3.2.3 Application server

The application server functions as an intermediary between the client, the deep neural networks, and the database together with the ground truth data. The mobile client can interact with the computer vision components through the back-end server by using the corresponding API endpoints. These requests include both training selected datasets and making predictions on new images just by pressing on buttons in the mobile app. These buttons are bound to the respective endpoints in my back-end server which then triggers the corresponding actions in the computer vision components. Eventually, the results from these actions are sent back to the mobile app so that they are available to be shown to the user. Another major functionality of my application server is the creation, storing, and editing of the application's data through the database. This is done

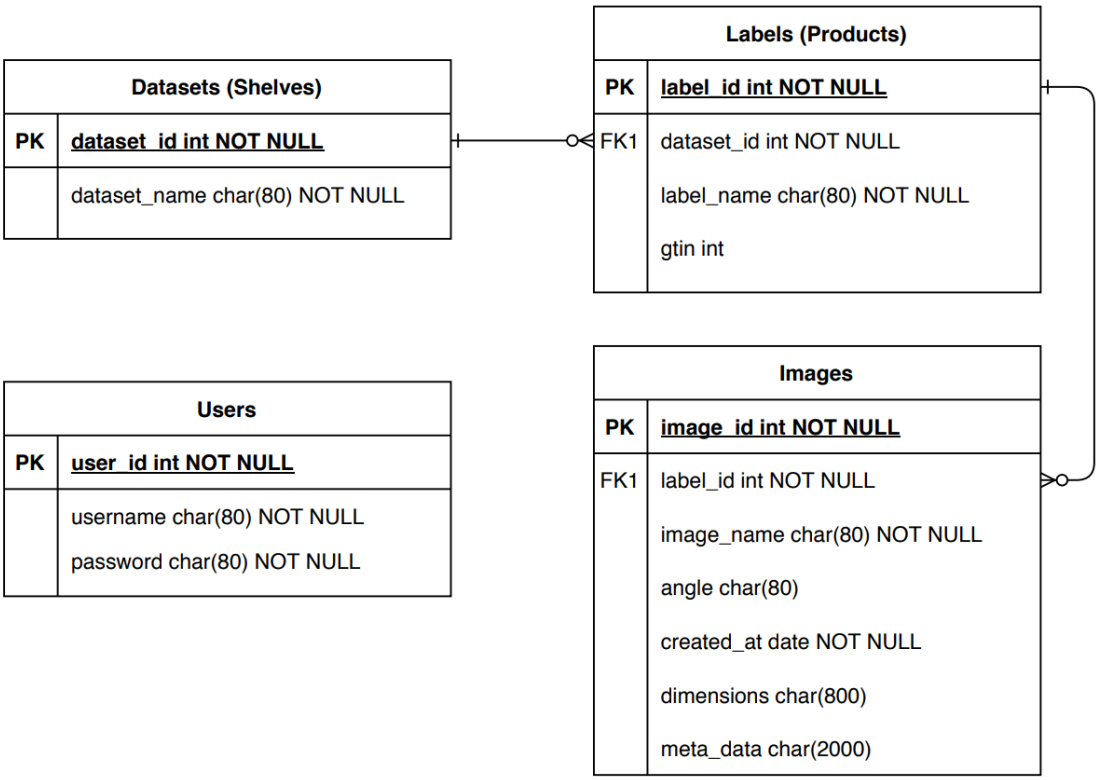



**Figure 3.3**: Entity relationship model diagram

by connecting the server with the SQL relational database management system which holds the entity data and the relationships among them. Moreover, the server is responsible for saving the photos taken from the mobile phone and serving them as static files so that they can be fetched and displayed in the mobile app.

### 3.2.4 External APIs

In addition, three external APIs are used; the Google Places API, EatFit Foodcoach API, and Open Food Facts API. With the help of Google Places API by providing the coordinates of the user's phone location we get the supermarket store where he is collecting the data and save this information as metadata together with the images. The store metadata can help group the products and datasets by store and then we can have a different deep neural network model for each store. When the conditions of the environment where your training datasets are collected match the real-world environment where the prediction will happen then the deep neural network models trained with this data result in better performance. The other two APIs are used to get the name of the product and create an entry for this entity in the database by using its barcode which is going to be used as the label for the image classification. Open Food Facts is a non-profit project with a database of over 1,448,268 food products developed by thousands of volunteers from around the world [30]. The EatFit Foodcoach API [31] is provided by Auto-ID Labs, an independent network of currently seven academic research labs, with a focus on swiss products and is also used for fetching the product information by its barcode. It is used as a complement for Open Food Facts

since when we scan a product's barcode, we check if the product exists in either one of the two APIs, and if not the user can manually enter the name of the product. This way the user doesn't always have to manually find out and enter the name of the product himself, rather he can just add the name by scanning its barcode.

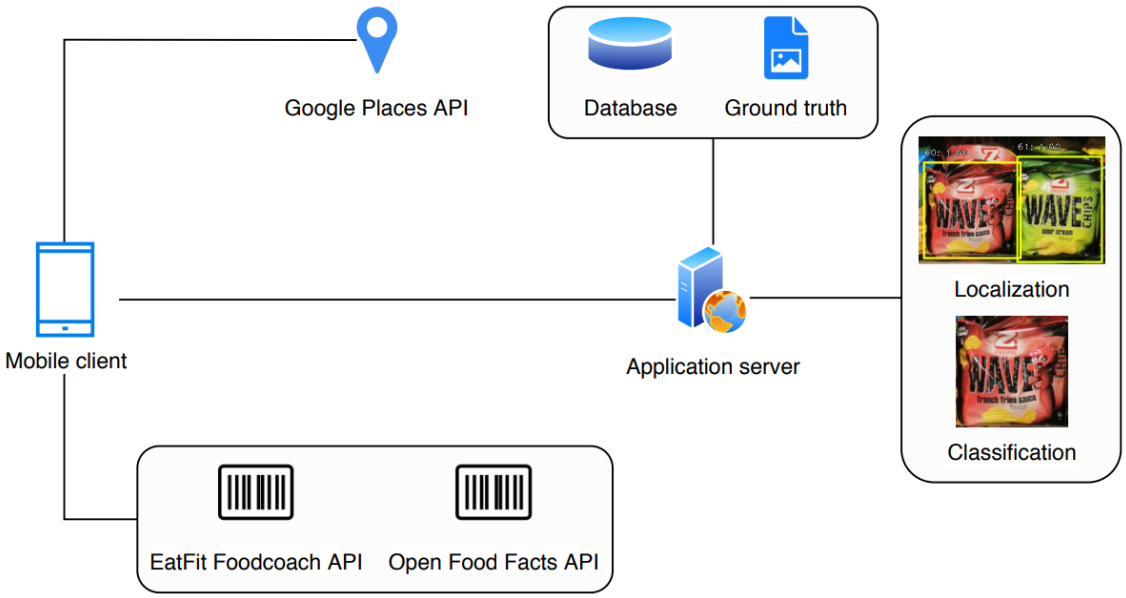



**Figure 3.4**: System overview

# Chapter 4

# Implementation

## 4.1 Front-end

In order to isolate parts of the app which have a specific purpose, and therefore code, we use React's composition in which we split the code into components.

### 4.1.1 Sign In Screen

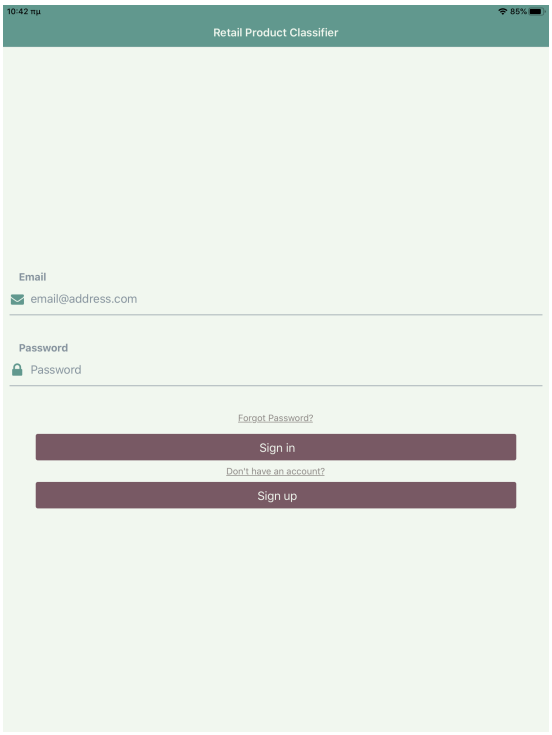



**Figure 4.1**: Sign In Screen

The purpose of this screen is to have user entities stored which can be used for a wide range of reasons. First, we can have different datasets for different users. Every user might be working on his own project which needs datasets specific to the topic he is working on. Collaboration can be boosted in case users want to invite other users to join their projects so that the training images for their datasets are generated faster. The user information is also stored as metadata for every image taken from the phone's camera. This information could be then used to see how valuable datasets a specific user generates. For example, a user generating bad quality datasets could possibly learn from another user generating good quality datasets and find out what he is doing wrong.

This is a pretty straight forward screen. The user can either sign in if he already has an account, or he can sign up by providing an email and password. Both sign in and sign up requests are handled by the back-end server where a user entity is created. For every user two JSON Web Token (JWT) tokens are created, an access token and a refresh token. The access token is used in the Authorization header of each request made by the mobile app to the back-end API and the server uses this token to decide if the client is authenticated or not. The refresh token is used to get a new access token in case the access token has expired, or a user wants to perform sensitive actions such as deleting a resource. After the user signs in the user data including the tokens are saved in the device so that the user will not have to sign in and get his JWT-tokens every time he wants to use the app. For this purpose, AsyncStorage API is used which is an asynchronous, unencrypted, persistent, key-value storage system for React Native to save application-related data locally in the mobile device.

### 4.1.2 Home Screen

The home screen serves as the start screen of the application for a logged-in user. It shows to the user an overview of the datasets that he has created. Since the main use of the app is to identify packaged products in supermarket shelves, we call the datasets shelves. The idea for using shelves is that the user can better structure his image collection by focusing on finishing each shelf separately. Consequently, we have a deep learning model for each shelf in the supermarket. However, the datasets can be named and grouped differently according to the preferred grouping of the datasets and their models. For instance, one could create a dataset for each retail store or even a mixed grouping of datasets. The database is developed in a generic way which gives the flexibility to the user to create and manage different kinds of datasets. For each dataset, we can see the number of labels, which in our case represent products, and immediately start either collecting labelled images for this dataset or perform inference by using an already trained model, if available. In addition, on this screen, we have the ability to create new datasets as well as navigate from the home screen to the settings screen.

When this screen starts for the very first time it will ask the user for some permissions needed for the application to be used correctly. These include the camera, location, and file system access permissions. Camera permission is needed to be able to use the camera for generating the image datasets. Location permission is used for getting the users location and, with the help of Google Places API, the store where he is collecting the dataset images at that moment. The location information is saved as metadata for each image taken. The datasets are shown as list items of the dataset list. Every dataset list item includes information for how many product labels it contains as well as whether there is already a trained model with this dataset. There are also two buttons on the right of the shelf list item called *Collect* and *Classify*. By pressing on the Collect button the user navigates to the Camera Screen used for collecting and generating photos for the dataset. The *Classify* button is only enabled if an already trained deep learning model exists for this dataset and is used for running inference on new images taken from the phone's camera. For creating a new dataset, a modal dialog is shown where the user can give the shelf name and send the request

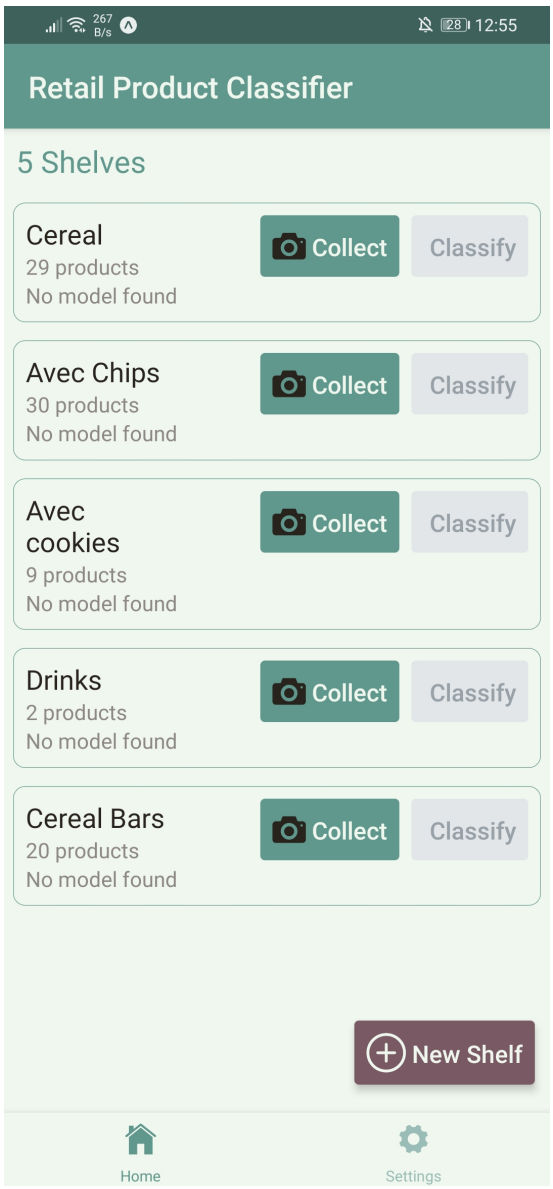



**Figure 4.2**: Home Screen

to the back-end server to create the dataset entity.

### 4.1.3 Dataset Screen

This screen provides a way to manage the products inside a given dataset. After a user presses on a dataset list item of the Home Screen he is then navigated to the Dataset Screen. Here we have an overview of the different product labels in this dataset in form of a list. The list can be refreshed at any time by swiping down from the top of the screen to get the latest state of the products in the database. For every product list item there is an image of the product shown, its title, and

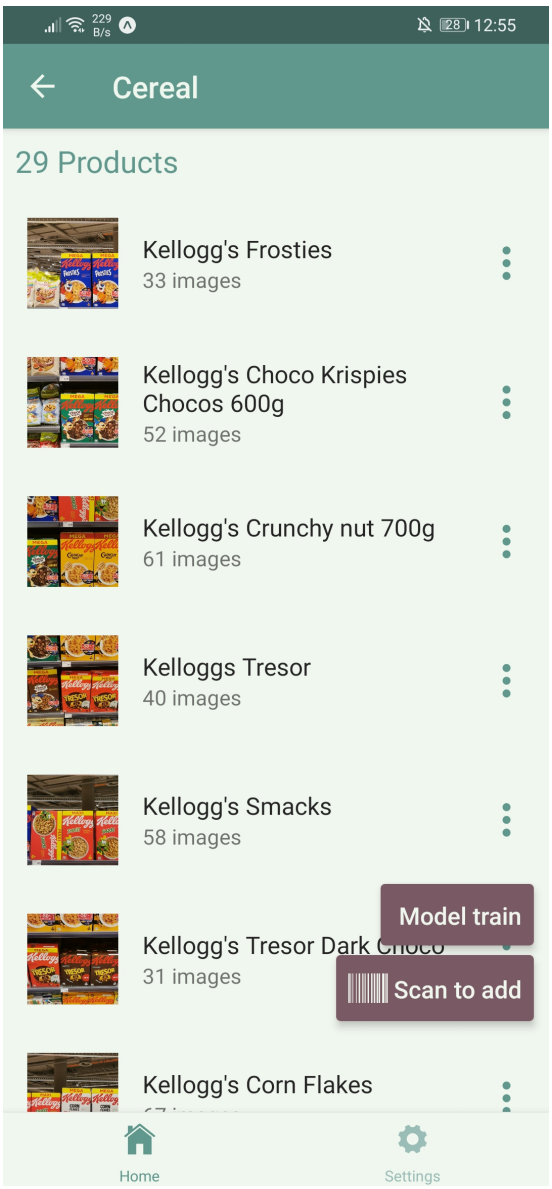



**Figure 4.3**: Dataset Screen

also how many training images exist for this specific product. On the right side of the product, we have an action menu in form of a popover where the user can delete this product. There are two ways in which the user is able to create a new product label. By pressing the *Manually Add* button a modal dialog opens where the user can add the product title of the product that he wants to add. When the *Scan to Add* button is pressed the user will be navigated to the camera screen where he can create a product by scanning its bar-code. As we already mentioned, two different APIs are used to fetch the product by its barcode, one offered by eatfit-service.foodcoa.ch and the other one by world.openfoodfacts.org. After the bar-code has been scanned a request is sent to the back-end server in order to create the product entity in the database.

### 4.1.4 Label Screen

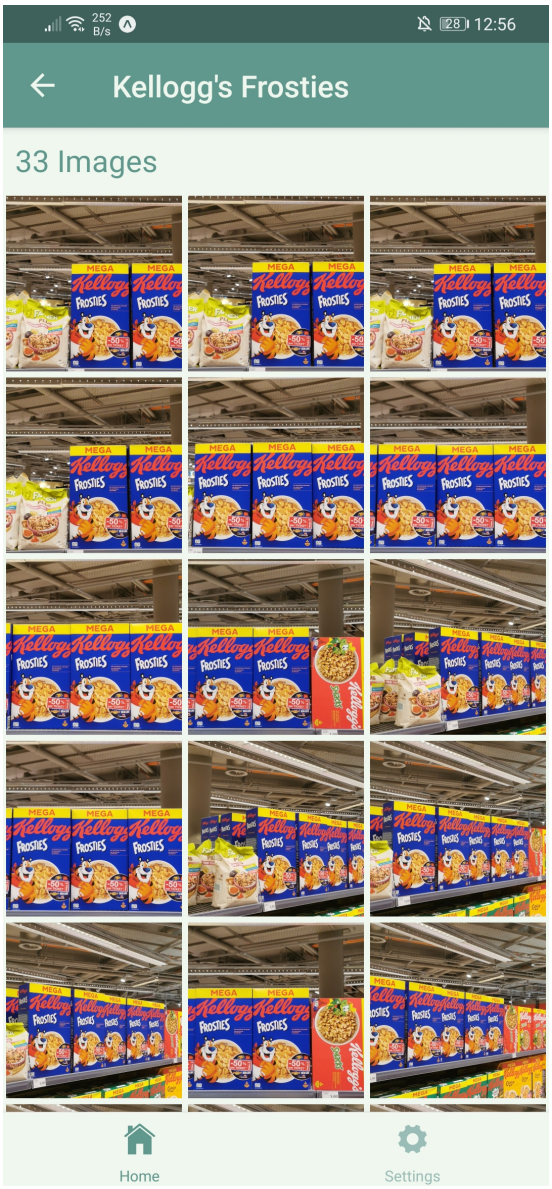



**Figure 4.4**: Label Screen

The Label screen is used for viewing the images taken of a specific product label. To navigate in this screen, you need to press on a product label list item from the Dataset Screen. For rendering the images, we use the FlatList component from React Native which uses virtualization, a fast and efficient way of rendering big lists where only the currently shown elements are rendered. Again, to fetch the most up to date list of images from the database, one can refresh the list at any time by swiping down from the top of the screen. The image list is split into three columns and the

user can press on an image in order to navigate to the Image Screen.

### 4.1.5 Image Screen

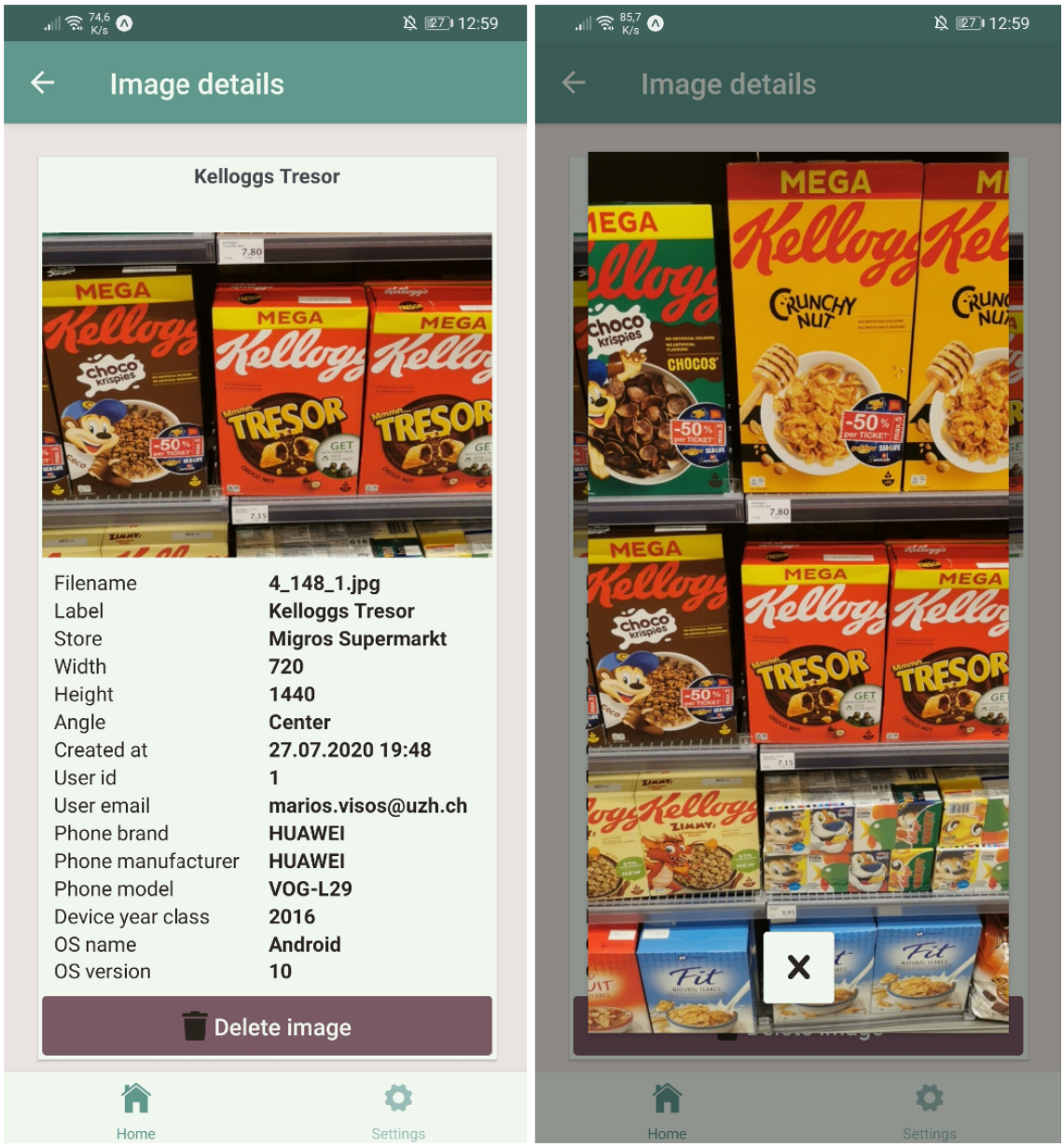



**Figure 4.5**: Image Screen

The image screen provides us with details about an image of a given product label in a dataset. Apart from the image itself, the user can see all the different properties and metadata saved along-

side the photo. The image properties include the filename, the creation date-time, the width and height of the image. As for the metadata, we have information about the device which was used to take the photo such as phone brand, model, manufacturer, device year class, the operating system version, and name. User related information such as the email and id of the user that took this photo is also displayed. Moreover, the angle, either left, right, or center, from which the photo was taken is provided together with the name of the label of the product in this image. Last but not least, the name of the retail store where the photo was taken is also shown.

### 4.1.6 Camera Screen


**Figure 4.6**: Camera Screen

The Camera Screen in the cornerstone screen of the mobile application. It is responsible for guiding the user through the whole process of generating meaningful labelled image datasets by using the phone's camera. Every image taken by the user includes precious information in form of metadata. First, the position of the bounding box is used as a label for the detection and localization of objects, in our case products, within images. Second, the image label is needed for training the deep learning model to learn and classify which product lies within each bounding

box. In addition, the information of the user who is taking the photos as well as his device's model are saved since they can be indicators of meaningless dataset generation.

As with the rest of the app, we also use React's composition model in this screen by splitting its logic and code into React components. There are five main components in this screen, the NearbyStores, the BarCodeScan, the BoundingBox, the ProgressBars, and the CameraTutorialOverlay.

### NearbyStores.js

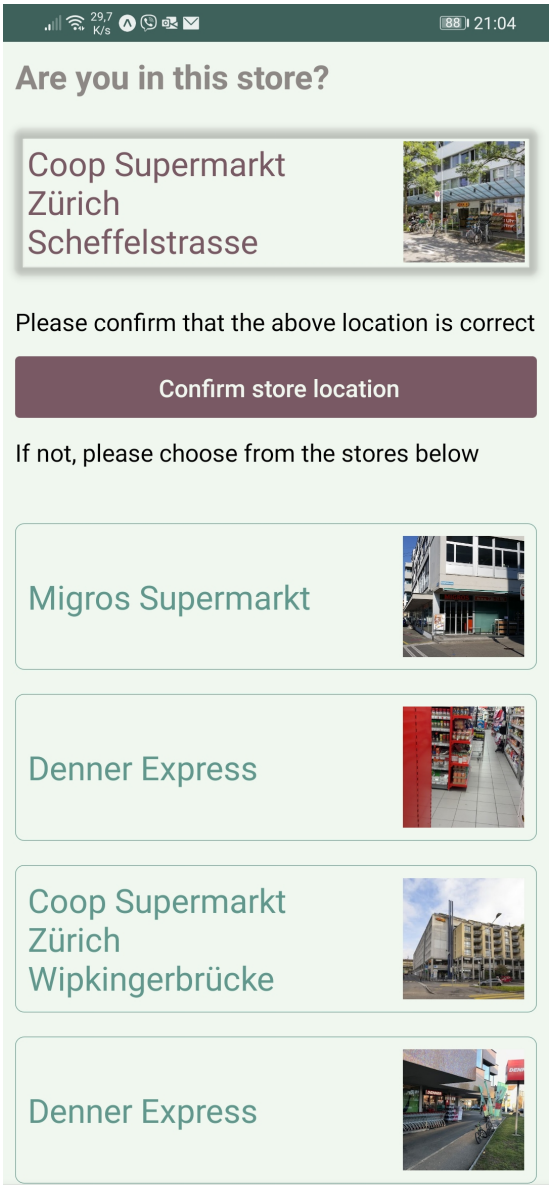



**Figure 4.7**: NearbyStores Component

The NearbyStores component asks the user to confirm the retailer's store in which he is at that moment. By using the geolocation information from the device in Google Places API we fetch the nearby supermarkets and show 20 stores in ascending order by their distance from the specified location. The user is then asked to either confirm the pre-selected closest found store or choose another one from the list. The stores are shown in form of a list where each store list item shows a photo of the store fetched by Google Places API as well as the store name.

### BarCodeScan.js


**Figure 4.8**: BarCodeScan component

The BarCodeScan component's main purpose is to get the product label by scanning its barcode and include it in every photo taken by the user. After the user has scanned the product's barcode a request is sent to my back-end server for fetching the product by its barcode. In case this product doesn't exist in our database we use 2 different APIs which provide endpoints for fetching a product by its barcode, the eatfit-service.foodcoa.ch, and the world.openfoodfacts.org. After the product label was fetched by either one of the APIs, we then proceed with a request to my back-end server for creating this entity in our database. If neither of the APIs contains that product, a modal dialog is shown to the screen which informs the user that the product is not found. In this modal dialog, the user is prompted to enter the product name manually in a text input field so that we then create the entity in our database.

## BoundingBox.js

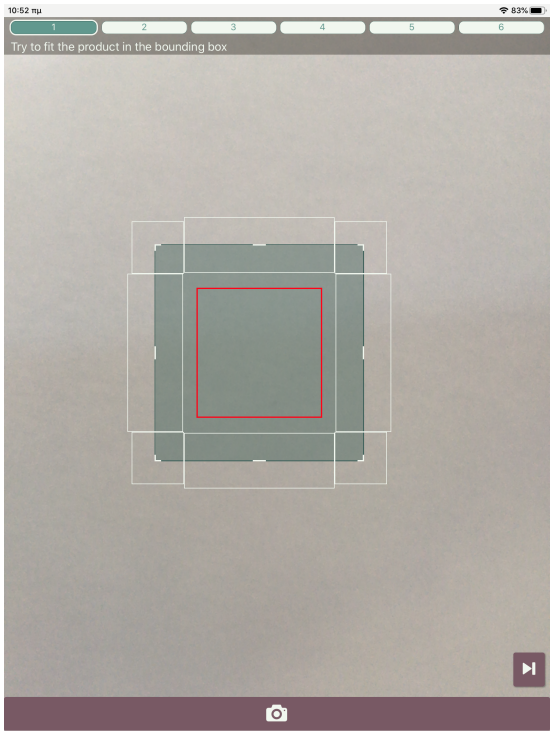



**Figure 4.9**: BoundingBox component

The BoundingBox component provides us with the ability to add and configure bounding boxes in our images. A bounding box is used as a label for the deep neural network model responsible for the detection and localization of products within a picture. A pan gesture happens whenever the user moves one or more fingers on the edge of the screen. As we can see in figure 4.9 by using nine pan gesture drag handlers the user can both resize and move the bounding box in the screen on top of the rendered preview of the device's back camera. Eight of these handlers are used for resizing the bounding box, 4 of which are located at the box's corners, and the other four are located at the box's edges. Apart from the resizing drag handlers, another drag handler responsible for moving the bounding box, while it is preserving its dimensions, was

implemented. The drag handler responsible for the drag and drop behaviour of the box is located in its center and takes up 60% of the area of our bounding box. The bounding box is developed in such a way that it is fully customizable and easy to configure. In the BoundingBoxProperties.js file located in the constants directory of our app, one can set predefined properties such as the minimum width and height, the area of each drag handler which defines how much space it takes on the screen for the user to be able to drag it as well as different style-related properties like color and border width. Another, remarkable fact about the bounding box is that since it is split into its own component we can use multiple bounding box components at the same time.

For every photo taken by the user, the bounding box information is saved in form of metadata. These include the top-left and bottom-right coordinates of the corners as well as the width and height of the box. Due to the fact that the bounding box is rendered on the phone's screen, its position and dimensions are proportionate to the screen's dimensions. Consequently, to have the corresponding position and dimensions in the actual image saved in our database there is a mapping used that takes into account the image and phone's screen dimensions. Moreover, since every phone has a different size for its screen and the photos taken from its camera the rule of the three is used to calculate the bounding box dimensions which correspond to the photo taken.

## 4.2 Back-end

### 4.2.1 Application server

A resource is a major concept of any RESTful API which relates to other resources and has a type and some methods that define its behavior [32]. Resources in our case are matched with our entities in my Entity-Relationship (ER) diagram and for every resource, there are data associated with them which represent the resource model of my API. These data are most of the time saved in a table of my SQL database or are saved as static files. Every resource in my API has its own absolute URL relative to my API's entry point which gives the ability to the client to interact with the API. The following sections are structured with respect to the resources as well as the model associated with them.

#### Dataset resource

The dataset resource represents a deep learning dataset of images that have certain annotations as well as models associated with them. As mentioned already, because this thesis is focused on the identification of packaged goods in the retail environment the datasets are renamed and displayed as shelves in the mobile application client. However, since this is just a naming convention one can create datasets according to his needs. The server exposes 4 URL endpoints of the dataset resource two of which are related to the dataset entity itself and the other two used for linking our server with the computer vision components. The first one can be accessed via three different HTTP methods, GET, POST, and DELETE. When a user wants to fetch a specific dataset by its name he can use the GET HTTP method and then our resource will return the JSON (JavaScript Object Notation) representation of it, if it exists in our database. Otherwise it returns a response with status code 404 which means that the dataset was not found. The POST HTTP method can be used by giving the dataset name and has two purposes. One is the creation of new dataset entities in our database and the other is the update of already existing ones. The DELETE method is used in order to have the ability to delete datasets. Because deletion is quite a sensitive action a decorator is used which requires the user to re-enter his password in order to generate a new JWT-token used for authentication. Additionally, we have one more endpoint containing only a GET method for fetching an array containing all the datasets ready to be displayed in the mobile

application in form of a list. The last two endpoints are about the deep neural network models which are associated with each dataset. Both these endpoints have only a GET method and are the way in which our application server is linked with our computer vision components. One of them is used for preparing all the data needed for training a model with a selected dataset and trigger the training once the data is in the desired format. The other endpoint is used for classifying new unseen images by using a trained model of a selected dataset and when the prediction is finished it sends the result back to the mobile app.

The dataset resource is persisted in the SQL database via the dataset model. Its properties include an id, a name as well as a list containing the label ids from the label entities it relates to. Furthermore, the dataset model object includes 5 methods three of which are used for filtering datasets, either by name or by id, and two which are used for saving or deleting database entity entries into or from the database.

### Label resource

The label resource represents the label including the annotations needed in order to have datasets ready to be fed in the training process of a deep learning model. Again in this case, as we also did with the dataset entity, in our mobile application client we use the term product instead of label. Nevertheless, this is just a naming convention for the sake of our application's specific domain. Our application is designed in a way that the label can be generic and be used for other domains as well. Our API exposes 2 URL endpoints coupled with the label resource. One of the endpoints is accessible by using the GET, POST, PUT and DELETE HTTP methods. The user can fetch a particular label by providing either its name or its barcode in the GET request and our API will return the JSON representation of the entity if it exists in our database. In our mobile application, the user can add a new label, which is going to be included in all the generated images, by scanning the product's barcode. As soon as the barcode is scanned the mobile app sends a GET request to the application server to see if we have already created an entity entry of the scanned product in our database. Therefore, instead of just fetching by name there was a need to be able to fetch by barcode so that we can prevent needless requests to external APIs for getting the product information by its barcode. The POST method is used in order to create new label entity entries in our database. This request is made from our mobile app either when the user wants to manually add a label by using a text input or it is automatically made after a product is fetched by an external API. This way when we fetch a product from an external API the first time, we automatically create an entry into our database so that we can avoid having to reach external APIs again for any future requests regarding this product. Additionally, we use the PUT method for updating a label and the DELETE method for deleting label entity entries from our database. Apart from the endpoint used for performing the basic CRUD (CREATE, READ, UPDATE, DELETE) actions for our entity, we provide a second endpoint for fetching a list of products that are used from the mobile client in order to display the products in form of a list.

A label model is used for managing the dynamic data persistence of label entities in our tabular SQL data structure. The label table consists of 4 columns, id, name, barcode, and dataset id. Besides, as we already saw in the ERM figure 3.3, it is linked with the dataset and the image entity tables since a dataset has multiple labels and a label has multiple images. Moreover, there are four methods for retrieving filtered lists of labels, filter by either name, barcode, id, or no filter for fetching all labels. Finally, I have two more methods, one for creating entity entries and another for deleting them from the database.

### Image resource

The image resource represents the images that are used in the training, validation, and testing phases of a deep learning model. There are three endpoints exposed related to the image resource two of which are used for performing basic CRUD operations and one for serving the static image files. Firstly, as with the rest of our entities, an endpoint can be reached by using one of the GET, POST, PUT, and DELETE HTTP methods. The GET method is used for fetching the image file which is exposed as a static file in the image folder. These images are displayed in the dataset screen where we have all the products within one dataset and we show for every list item an image of this product. The image files are also accessed in the label screen where we show all the images that we have collected for this product as well as in the image details screen where the user can also optionally view the image in full screen. In our image resource, there are two different ways in which we persist our images which complement each other. On the one hand, we have the image entity which is saved in our SQL database and includes all the information about an image and its relationship with the other entities. On the other hand, we have the actual image files which are saved in our desired image directory. A saved image can either be served as a static file, by using the image name saved in our database for getting the file path or be used as an input in a deep learning model.

As a result, the POST method of our image upload endpoint which is used for creating new images is split into two parts, one for creating the image entity and the other for saving the image file. For every photo taken by a user's phone in the camera screen of our mobile app, a POST request is made to our server which includes the image together with useful metadata, such as image properties, the angle from which the photo was taken, device information, retailer's store, and bounding box location. Our server handles this request by storing an image entry in our database including all its metadata as well as saving the file in our directory from which this image is going to be served. At this point, the initial implementation was to create all the additional files needed for the training process of the deep learning model to begin. In the current implementation, these files are not created every time we upload an image since it will make both the saving of the image and the response of the image resource slower. Alternatively, the files required for the training function are only generated when the training process is started and does not significantly affect the training duration since the process itself is slow.

This two-folded way of persisting the image data affects the PUT and DELETE methods as well, in a way that we need to update both the static files and the image entity entry in our SQL database table. The last endpoint related to the image resource has only a GET method and is used for fetching an array of images. One can either fetch all the existing images or he can optionally provide a label id so that he can only receive images of a certain product. This endpoint is used from the mobile app client to display the images of a product in a certain dataset in form of a list and also for showing the statically served image file.

The image model is backing up our image resource and serves as an intermediary between our API and the actual database. The image table has 7 columns including id, name, angle creation date, dimensions, label id as well as metadata. We have four methods used to filter the entries of our table by either name, id, label id, or without any filter for getting all the entries. Finally, we got a method for getting a JSON representation of an image and two methods for saving and deleting images in the SQL database table.

### User resource

Our last resource is associated with the users of our app and has multiple purposes. First of all, the existence of users allows them to generate and manage their own datasets according to their work and needs. We want our app to embrace and enable collaboration between individuals so that multiple groups, such as research groups, can create and have control over datasets specific

to the project they are working on. The generation of project-specific datasets can be faster when you invite more people to work on it. Dataset quality is one of our main concerns and by saving the user information together with other metadata in our datasets we can assess how meaningful are the models trained with specific datasets. One can train models with datasets generated from certain users and examine what settings they used and how their datasets were generated so that he can judge the criteria which affect his model's performance and accuracy. Our user resource is also strongly coupled with the authentication of our application by having JWT tokens per user.

There are six endpoints related to our user resource. The first one is about registering new users and it has one POST method. This method accepts requests including the username and password of a new user and uses our user model in order to save the newly created user in our database. If the username already exists then it sends an error back in the response so that the user chooses a different unique username.

The second endpoint is used for signing in a user and has a POST method. Requests sent to this endpoint include a username and an encrypted password. The username is used for finding the user in our database and the password is compared with the stored user password to see if it matched for proceeding with the login. Once the credentials match, two tokens are generated with the user's id, an access token, and a refresh token. The access token is included in the authorization header of all the requests made from this user from the mobile client. The refresh token is needed for getting a new access token and is useful for cases when an access token has expired or for sensitive actions such as deletion.

After a user has signed up and logged in, he should also be able to log out. Therefore, a user logout endpoint is also implemented with a POST method used for signing out a user. In this method, we get the id of the JWT which is included in the header of the user's logout request, and add this to a blacklist so that it cannot be used anymore in the future and a new one will have to be generated.

Moreover, we have a user endpoint that has two methods. A GET method is used for fetching a user from the database using the user model. The second method is a delete method and is used for deleting users from our database. The fifth endpoint related to the user is for resetting his password. A POST method is implemented in which the user includes his email address and an email is sent to him in order to be able to reset his password.

Last but not least, there is an endpoint for refreshing a JWT-token. In the POST method of this request, we receive the refresh token included in the authorization header of the user's request and use it in order to generate a fresh access token. This access token can then be used from the user for authorization until it either expires, or the user logs out or if he needs to perform a sensitive action and he will have to generate a new one.

### 4.2.2 Computer vision

In our back-end architecture, the communication between the server and the computer vision components is designed in a modular way following the separation of concerns design principle. Our computer vision part consists, on a high level, of two main components, object detection, and image classification. Thereby, the code regarding these components is abstracted and is only accessed through two endpoints in our API one for training and the other for predicting. This allows individuals to use our app and integrate it with their preferred pipelines for the part regarding the deep neural networks. The only thing one would have to do to use our app with his preferred computer vision architecture is to write a utility function that would convert the annotations which are already included in our database into the format of his desired algorithms. To demonstrate this separation, our software system was prepared to be tested with three different computer vision components, which are also explained in more detail below, one for object localization, the second for image classification, and a pipeline including both object localization and

image classification at the same time.

## Object localization

For the object localization, we use the algorithm proposed by Goldmann et al. [1]. Intuitively this algorithm, as we already saw in section 2.2, simplifies the original mixture of many Gaussians into a mixture of few distinct Gaussians while preserving the same structure of the entire distribution. This is applied with expectation-maximization which is done iteratively, followed by an expectation step (see equation 2.16) which minimizes the distance between the mixture of original Gaussians and the mixture of new Gaussians. Finally, the maximization step (see equation 2.17 2.18 2.19) estimates the parameters of the mixture of new Gaussians.

As we already mentioned, both training and prediction algorithms of the object localization component are accessed through two endpoints exposed by our server. In order to be able to run the training of the object localization model, we need to generate the required annotation files in the desired format. These include three (Comma Separated Value) CSV files, all in the same format corresponding to the train, validation, and test data respectively as well as the actual image files. As we can see in table 4.1 containing the format of the CSV files, we have 8 columns and every row represents a bounding box annotation in an image. The first column contains the name of the image file which is used in order to access the file from the static folder where it is saved. The following 4 columns include the bounding box coordinates within the image, two of which are used for the top-left corner coordinates, and the other two for the x and y coordinates of the bottom-right corner of the box. The sixth column represents the class and is set to *"object"* for all the instances since we only care about object localization, which just locates objects in an image without classifying an image in a label. The last two columns correspond to the width and height of the image, the location of which is mapped to the folder directory in which is saved and served as a static file by our application server.

On a high level, the process of generating the required files needed for the training algorithm is the same regardless of the CNN architecture used. When a user triggers a training process he selects a dataset that we then fetch from our database. The only change needed in order to train another algorithm is the creation of a utility function that will convert the annotation information from our database into the desired format. These utility functions act as a compatibility layer between our application server and the desired deep neural network. As shown in the code listing 4.1 by looping through all the labels of that dataset and its images, we create an annotations CSV file which is then split into three files one for each of the train, validation, and test phases. Besides, as you can see in the code listing 4.2 we have a second utility function that splits the annotations.csv that we created into the train, validation, and test files. Once all the files are created, we then run the code for training our object detector with our generated data.

| name | top_left_x | top_left_y | bottom_right_x | bottom_right_y | class | width | height |
|---|---|---|---|---|---|---|---|
| 34_1210_1.jpg | 687 | 1337 | 1224 | 1944 | object | 2268 | 4032 |
| 29_1640_2.jpg | 200 | 857 | 340 | 1049 | object | 720 | 1440 |
| 38_1285_1.jpg | 240 | 1061 | 352 | 1272 | object | 720 | 1440 |
| 57_1763_3.jpg | 496 | 672 | 658 | 898 | object | 720 | 1440 |
| 25_1085_1.jpg | 260 | 625 | 419 | 814 | object | 720 | 1440 |
| 44_1403_3.jpg | 858 | 1755 | 1354 | 2535 | object | 2268 | 4032 |
| 40_1306_1.jpg | 893 | 1767 | 1333 | 2373 | object | 2268 | 4032 |
| 60_1591_1.jpg | 260 | 492 | 423 | 683 | object | 720 | 1440 |
| 26_1095_1.jpg | 175 | 610 | 335 | 832 | object | 720 | 1440 |
| 25_1084_3.jpg | 260 | 625 | 414 | 884 | object | 720 | 1440 |
| 26_1118_2.jpg | 89 | 574 | 200 | 757 | object | 720 | 1440 |
| 41_1320_1.jpg | 858 | 1775 | 1410 | 2258 | object | 2268 | 4032 |
| 54_1519_3.jpg | 858 | 1755 | 1243 | 2243 | object | 2268 | 4032 |
| 44_1392_1.jpg | 858 | 1755 | 1410 | 2277 | object | 2268 | 4032 |
| 43_1352_1.jpg | 194 | 1138 | 328 | 1329 | object | 720 | 1440 |
| 36_1255_3.jpg | 413 | 1056 | 524 | 1279 | object | 720 | 1440 |
| 34_1227_1.jpg | 790 | 1453 | 1182 | 1943 | object | 2268 | 4032 |
| 63_1789_1.jpg | 420 | 327 | 547 | 535 | object | 720 | 1440 |
| 57_1752_3.jpg | 488 | 811 | 629 | 1028 | object | 720 | 1440 |

**Table 4.1**: Table containing the CSV data format for the annotation file used for training the object localization model

```
def generate_annotations:
   with open('annotations.csv', 'w', newline='') as csvfile:
      # Loop through labels
      for counter, label in enumerate(labels):
         images = ImageModel.find_by_label_id(label.id)
         # Loop through images
         for image in images:
            meta_data = json.loads(image.meta_data)
            dimensions = json.loads(image.dimensions)
            bounding_box = meta_data['bounding_box']
            # Write the row in the annotations.csv file
            annotationswriter = csv.writer(csvfile)
            annotationswriter.writerow(
               [
                  image.name,
                  round(bounding_box['top_left']['x']),
                  round(bounding_box['top_left']['y']),
                  round(
                     bounding_box['bottom_right']['x']
                  ),
                  round(
                     bounding_box['bottom_right']['y']
                  ),
                  "object",
```

```
                dimensions['width'],
                dimensions['height']
            ]
        )
```

**Listing 4.1**: Utility function for converting our database annotations into a CSV needed for the object localization training algorithm.

```
def train_val_test_split():
   with open('annotations.csv') as annotations_file, \
         open(f'{folder}/annotations_train.csv',
             'w',
             newline=''
             ) as train_file, \
         open(f'{folder}/annotations_val.csv',
             'w',
             newline=''
             ) as val_file, \
         open(f'{folder}/annotations_test.csv',
             'w',
             newline=''
             ) as test_file:
      annotations_reader = csv.reader(annotations_file, delimiter=',')
      train_writer = csv.writer(train_file)
      val_writer = csv.writer(val_file)
      test_writer = csv.writer(test_file)
      # Define the train val test ratio.
      # In our case 80% - 10% - 10%
      train_ratio = 0.8
      test_ratio = 1 - train_ratio
      annotations = list(annotations_reader)
      annotations_length = len(annotations)

      # Randomize the data
      random.shuffle(annotations)

      num_of_test = int(math.ceil(test_ratio * annotations_length))
      num_of_val = num_of_test
      num_of_train = int(math.ceil(train_ratio * annotations_length))
      annotations_file.seek(0)
      test = []
      val = []
      train = []
      # Loop through annotations and write on the csv files
      for counter, row in enumerate(annotations):
         if counter + 1 <= num_of_test:
            test_writer.writerow(row)
            test.append(row)
         elif counter + 1 <= num_of_val * 2:
            val_writer.writerow(row)
            val.append(row)
         else:
            train_writer.writerow(row)
            train.append(row)
```

**Listing 4.2**: Utility function for splitting the annotations into train validation

Again regarding the prediction phase we expose an endpoint in our API for triggering the image reference. The mobile client includes an image in the request, on which we run inference, and is saved in a folder used by the object localization prediction algorithm. We then trigger the prediction algorithm which dumps the results in a folder statically served from our application server. These results consist of an image with the bounding boxes drawn around each detected object and a CSV file containing all the predicted bounding boxes in the same format as we saw in table 4.1. Consequently, the mobile app can have access to the results either in form of an image or just by using the annotation data to visualize the results by itself.

## Image classification

In order to better demonstrate our software's system compatibility with different deep neural network architectures, we chose quite a special image classification pipeline by Hoffer & Ailon et al. [29]. In contrast to the traditional image classification data which have one label included per image, we organize the data in triplets of images. Let's define $f(x)$ the embedding which embeds an image $x$ and the triplet loss motivated in Weinberger, Blitzer & Saul et al. [33] nearest-neighbor classification. Our goal is to make sure that an anchor image of a specific product is closer to all the other positive images of the same product than it is to any negative image of a different product and is nicely visualized in figure 4.10. Therefore, as they well defined in the work of Schroff et al. [2], we want,

$$||f(x_i^\alpha) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^\alpha) - f(x_i^n)||_2^2 \tag{4.1}$$

$$\forall (f(x_i^\alpha), f(x_i^p), f(x_i^n)) \in \mathcal{T} \tag{4.2}$$

where $\alpha$ is a margin enforced between positive and negative pairs and $\mathcal{T}$ is the set of all possible triplets in the training set with a cardinality $N$. They then define the loss as:

$$\sum_i^n [||f(x_i^\alpha) - f(x_i^p)||_2^2 - ||f(x_i^\alpha) - f(x_i^n)||_2^2 + \alpha]_+ \tag{4.3}$$

The generation of all the possible triplets would lead in too many triplets that are easily satisfied by fulfilling the constraint in equation 4.3. Therefore, since they would not contribute to the training and would make the convergence slower it is important that we select triplets which are active and can have a bigger contribution to our network's performance. As a result, we chose a specific approach for the triplet selection described below.

In order to speed up the training process, we need a smart selection of triplets. Since for every image, our mobile client application saves the angle from which the photos are taken, we use the angle information for generating meaningful training data. Thus, we generate triplets so that the anchor image and the positive image are taken from a different angle and the negative image is taken from the same angle as the anchor. On the one hand, by having the anchor and the negative images from the same angle we have similar looking images of different products. On the other hand, by pairing anchor and positive images from different angles we generate triplets of non-similar looking images for the same product. With this approach, we end up having less training data with more valuable learning information for our network.

As with every deep neural network architecture one wants to use with our software system, in this case, we also developed a utility function as a compatibility layer between the training of our model and our application server. As we can also see in the code listing 4.3, for every image we generate two triplets both having a similar angle for the anchor and negative samples and different angles for the anchor and positive samples. The generation of these triplets followed by the triggering of the training process can be accessed via a training endpoint of our application server API.

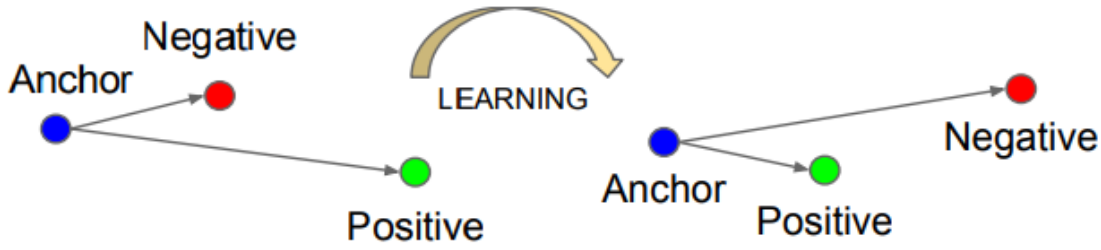



**Figure 4.10**: The Triplet Loss tries to minimize the distance between an anchor and a positive, both of which are from the same product, and maximizes the distance between the anchor and a negative of a different product [2]

```
angles = ['1', '2', '3']
test_ratio = 0.2
def get_negative_indexes(labels_by_angle, label_id, angle):
   '''Returns two negative indexes of the same angle'''
   rest_label_ids = list(labels_by_angle.keys())
   rest_label_ids.remove(label_id)
   label_id_1, label_id_2 = choices(rest_label_ids, k=2)
   negative_index_1 = choice(labels_by_angle[label_id_1][angle])
   negative_index_2 = choice(labels_by_angle[label_id_2][angle])
   return (negative_index_1, negative_index_2)
def get_positive_indexes(label, angle):
   '''Returns two positive indexes of different angle'''
   angle_1 = None
   angle_2 = None
   if (angle == '1'):
      if len(label['2']) > 0:
         angle_1 = '2'
      if len(label['3']) > 0:
         angle_2 = '3'
   if (angle == '2'):
      if len(label['1']) > 0:
         angle_1 = '1'
      if len(label['3']) > 0:
         angle_2 = '3'
   if (angle == '3'):
```

```
        if len(label['1']) > 0:
            angle_1 = '1'
        if len(label['2']) > 0:
            angle_2 = '2'
    positive_index_1 = choice(label[angle_1])
    positive_index_2 = choice(label[angle_2])
    return (positive_index_1, positive_index_2)
def generate_triplets():
    '''Generates two triplets for every image.'''
    with open('all_train_files.txt', 'r') as all_files_obj, \
            open('sampled_train_triplets.txt',
                'a',
                newline=''
                ) as train_obj, \
            open('sampled_test_triplets.txt',
                'a',
                newline=''
                ) as test_obj:
        images, labels_by_angle, num_of_test = group_labels_by_angles(
            all_files_obj)
        for (counter, image_name) in enumerate(images):
            anchor_index = counter

            label_id, image_id, angle = image_name.split('_')

            negative_index_1, negative_index_2 = get_negative_indexes(
                labels_by_angle, label_id, angle
            )

            label = labels_by_angle[label_id]
            positive_index_1, positive_index_2 = get_positive_indexes(
                label, angle
            )
            if (counter + 1 <= num_of_test):
                triplet_1 = (
                    f'{anchor_index} '
                    f'{positive_index_1} '
                    f'{negative_index_1}'
                )
                test_obj.write(triplet_1)
                test_obj.write('\n')
                triplet_2 = (
                    f'{anchor_index} '
                    f'{positive_index_2} '
                    f'{negative_index_2}'
                )
                test_obj.write(triplet_2)
                test_obj.write('\n')
```

```
else:
    triplet_1 = (
        f'{anchor_index} '
        f'{positive_index_1} '
        f'{negative_index_1}'
    )
    train_obj.write(triplet_1)
    train_obj.write('\n')
    triplet_2 = (
        f'{anchor_index} '
        f'{positive_index_2} '
        f'{negative_index_2}'
    )
    train_obj.write(triplet_2)
    train_obj.write('\n')
```

**Listing 4.3**: Utility function for generating triplets.

As we also did with the object localization architecture mentioned earlier, the same endpoint which is exposed from our API for predicting can be used in this case for classifying new unseen images. The input images for both training and prediction of the image classification should only contain one instance of the object of interest. Depending on the application, one can either collect and classify images with always one instance of the object or he can perform object localization for finding the location of the bounding boxes. Once one has the bounding boxes around the objects, by cropping them and getting the contained image we get from one image multiple images with just one instance of the object.

## Object detection

Object detection is the combination of both object localization and image classification at the same time. The two computer vision components described in the previous sections comprised of either object localization or image classification. If one wants to perform object detection he should perform object localization just for finding the objects and for each detected object he would need to perform image classification. Since the approach we used from Goldmann et al. [1] for object localization is built on top of Lin et al.'s [3] model which supports object detection (localization and classification at the same time) we also tested our software system using their implementation.

Again, we use the two aforementioned endpoints of our API exposed for training and prediction. As one can see in table 4.2, the desired format of the annotation files for object detection is almost identical to the one used in the object localization. The only difference is in the 6th column which contains the name of the class where just one class is used for the object localization while for the object detection we use the id of the object entity in our database.

The required files which are needed for training the object detection are identical to the one we used in the object localization. Similarly, the user selects a dataset of products that he wants to feed in a model for training. Likewise, the code that we saw in the listing 4.1 is also used for iterating over all the labels of that dataset and its images and creating an annotations CSV file which is then split into train, validation, and test files. The only difference in the code is that the class id is added in the class value of the CSV file instead of just one object class and an extra generated file containing a mapping of the class labels to ids. Besides, because the format of the files needed is similar, the utility function in code listing 4.2, which splits the annotations.csv that we created into train, validation, and test files, remains the same. As soon as all the files are

| name | top_left_x | top_left_y | bottom_right_x | bottom_right_y | class | width | height |
|---|---|---|---|---|---|---|---|
| 10_441_1.jpg | 394 | 570 | 627 | 935 | 10 | 720 | 1440 |
| 10_480_2.jpg | 276 | 513 | 365 | 716 | 10 | 720 | 1440 |
| 6_247_1.jpg | 260 | 625 | 502 | 919 | 6 | 720 | 1440 |
| 7_322_1.jpg | 262 | 526 | 438 | 813 | 7 | 720 | 1440 |
| 10_497_3.jpg | 276 | 513 | 387 | 722 | 10 | 720 | 1440 |
| 2_56_3.jpg | 404 | 656 | 558 | 968 | 2 | 720 | 1440 |
| 23_1062_2.jpg | 260 | 625 | 358 | 782 | 23 | 720 | 1440 |
| 15_719_1.jpg | 446 | 616 | 598 | 806 | 15 | 720 | 1440 |
| 3_97_2.jpg | 346 | 516 | 555 | 984 | 3 | 720 | 1440 |
| 13_616_2.jpg | 176 | 572 | 334 | 878 | 13 | 720 | 1440 |
| 10_455_2.jpg | 465 | 544 | 573 | 817 | 10 | 720 | 1440 |
| 4_161_2.jpg | 436 | 594 | 571 | 874 | 4 | 720 | 1440 |
| 13_603_1.jpg | 260 | 625 | 495 | 962 | 13 | 720 | 1440 |
| 18_833_2.jpg | 260 | 625 | 393 | 835 | 18 | 720 | 1440 |
| 19_883_3.jpg | 260 | 625 | 378 | 836 | 19 | 720 | 1440 |
| 7_278_1.jpg | 260 | 625 | 517 | 1037 | 7 | 720 | 1440 |
| 12_578_1.jpg | 259 | 625 | 420 | 862 | 12 | 720 | 1440 |
| 5_189_1.jpg | 260 | 625 | 534 | 1030 | 5 | 720 | 1440 |
| 7_319_1.jpg | 262 | 526 | 438 | 813 | 7 | 720 | 1440 |
| 2_35_1.jpg | 260 | 625 | 529 | 1036 | 2 | 720 | 1440 |

**Table 4.2**: Table containing the CSV data format for the annotation file used for the object detection model

ready in the desired format, we then run the code for training the object detection model with our generated data.

As for the prediction, we use an exposed endpoint of our API so that the mobile client can trigger the inference via an HTTP request. In that request, an image is included for running the prediction which we save in a folder accessed by the object detection algorithm. The prediction results are two-folded. One part includes a CSV file, the format of which can be seen in table 4.3, with 7 columns. The other part includes the image with the bounding boxes and the labels that are drawn on the same image around the detected objects. The output image is saved in a folder that is statically served by the application server in order for the mobile client to be able to access and show it to the user.

| name | top_left_x | top_left_y | bottom_right_x | bottom_right_y | confidence | class_id |
|---|---|---|---|---|---|---|
| 47_1677_1.jpg | 601.29803 | 1037.213 | 702.84705 | 1246.4928 | 0.9600425 | 8 |
| 47_1677_1.jpg | 575.2071 | 238.25307 | 698.3076 | 431.80045 | 0.978906 | 18 |
| 47_1677_1.jpg | 170.40843 | 890.59406 | 322.5624 | 1063.9568 | 0.98636174 | 4 |
| 47_1677_1.jpg | 306.61545 | 409.65823 | 521.901 | 630.8325 | 0.9008447 | 27 |
| 47_1677_1.jpg | 405.9247 | 581.60657 | 624.4672 | 808.26105 | 0.99999714 | 18 |
| 47_1677_1.jpg | 417.35056 | 578.5825 | 647.365 | 810.93414 | 0.98497795 | 5 |
| 47_1677_1.jpg | 408.45117 | 584.86786 | 624.0605 | 807.61597 | 0.97969984 | 28 |
| 47_1677_1.jpg | 407.0588 | 585.6409 | 623.1355 | 805.30835 | 0.94823164 | 0 |

**Table 4.3**: Table containing the CSV data format of the inference output file of the object detection model

# Chapter 5

# Evaluation

A usability study was performed in order to evaluate the usefulness of our application and get valuable feedback from users. This would help us answer our first research question: *Can a mobile application reinforce a faster and more meaningful generation of labelled Convolutional Neural Network datasets?* Since images collected under lab conditions do not always match the real-world environment, we performed the study in a real supermarket store with the goal to collect images for training which are as close as possible to the real-world environment. After the usability study, we selected a dataset generated by the users and fed it in the computer components mentioned in Section 3.2.2 in order to answer our second research question: *Is a mobile application sufficient for replacing the manual creation and annotation of datasets for both object localization and object classification while having similar accuracy?*

## 5.1 Study settings

The evaluation was performed with a total of fourteen participants in one session for each one of them over a period of two weeks. For the purpose of this study, we were granted permission from an authorized employee of the Swiss company called Valora in order to be able to conduct the study in one of their supermarket stores named "avec" located at the Zurich main station. The users were given an explanation of how the manual collection and annotation of datasets work in case they were not aware of the procedure so that they can compare the manual process to the one offered from us. Both the development server of our mobile application client and our application back-end server were exposed through a library called ngrok, a reverse proxy that exposes web service running locally to a public endpoint by creating a secure tunnel [34]. Consequently, the users could run the app on their mobile phones since they could access both the mobile app served by a development server and the back-end server through an internet connection. This allowed us to generate data collected from different devices with different cameras.

After they had used the mobile application, they were asked to complete a feedback questionnaire (see figures 5.1, 5.2) consisted of 6 questions. Three of the questions used the seven-point Likert scale and the other three could be answered via short text input.

# Master Thesis Retail Product Classifier

1. How likely is it that you use the app in the future for your project?

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Not at all | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | Very likely |

2. How likely is it that you recommend the application to someone?

*Mark only one oval.*

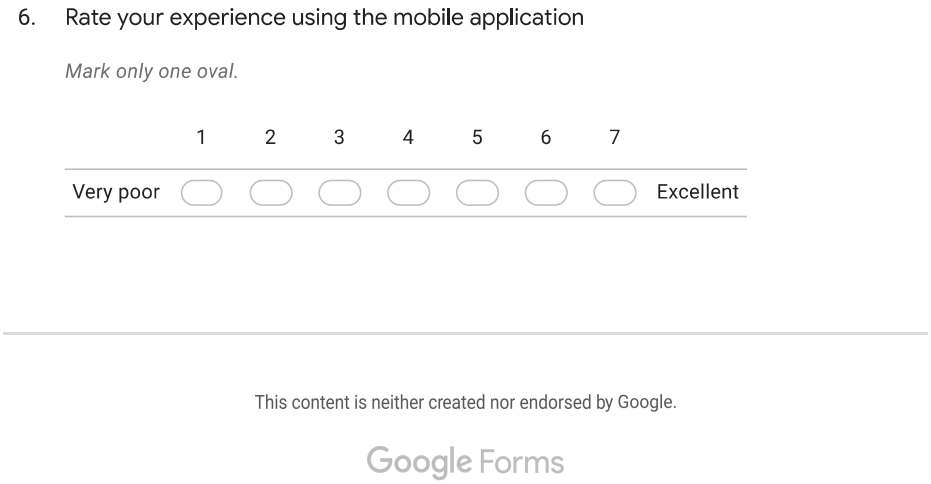
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Not at all | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | Very likely |

3. What did you like most about the mobile application?

4. What did you like least about the mobile application?

5. What do you think the mobile application should improve on?

**Figure 5.1**: Questionnaire first page


6. Rate your experience using the mobile application

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Very poor | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | Excellent |



**Figure 5.2**: Questionnaire second page

# 5.2 Results

## 5.2.1 Dataset

The users generated 4 datasets including 88 product labels and 2630 images with 3257 bounding boxes. Moreover, a dataset associated with one supermarket shelf was chosen, in order to be fed in the object localization and object detection computer vision components mentioned in the Subsection 3.2.2, for evaluation. The dataset is a shelf of chips containing 537 images and 537 bounding box annotations including product labels. Finally, as we have one bounding box per image we manually annotated the rest of the instances resulting in 28 of the 537 images selected from different angles with a total of 655 annotations.

## 5.2.2 Usability study

### Linear-scaled questions

As we already mentioned we had three Likert-scaled questions. One of them asked the user to rate the overall experience using our application. The other two questions asked specifically about the willingness of a user to use this app for their next project and if they would recommend the application to someone. The user responses of these questions can be seen in the histograms 5.3, 5.4 and 5.8.

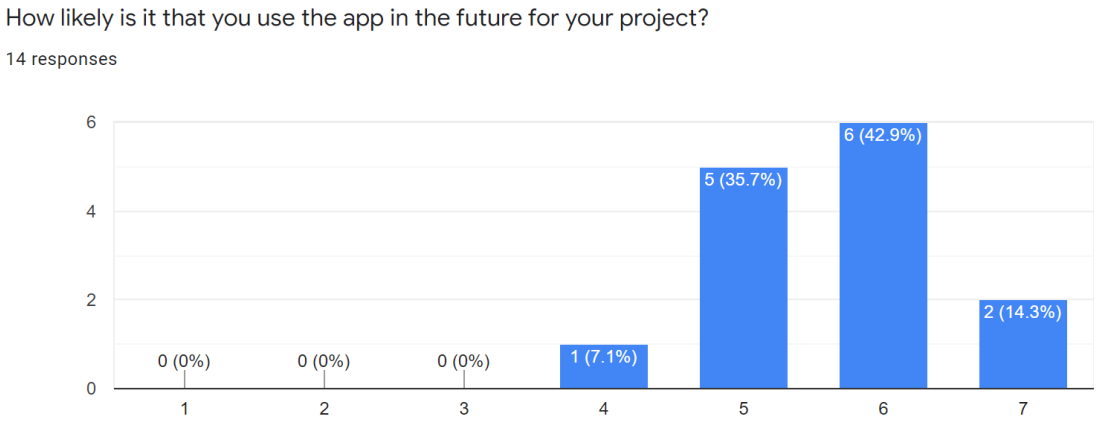



**Figure 5.3**: The results of the first question

### Short text answered questions

In addition to the linear scale answered questions we included three questions that could be answered via free text input. Two of them were about the most liked as well as the least liked feature of the mobile application. The answers to these questions can be seen in figures 5.5, 5.6, 5.7.

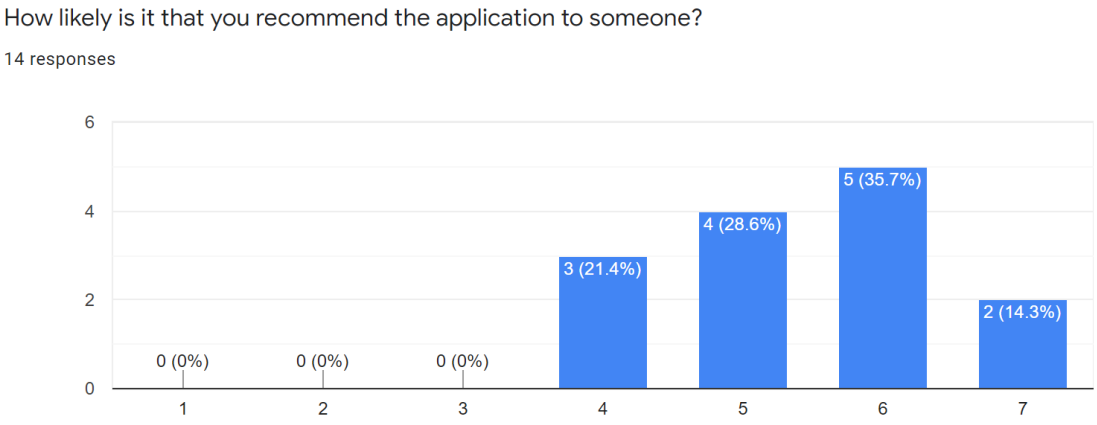



**Figure 5.4**: The results of the second question

### 5.2.3 Dataset evaluation

#### Object localization

In this section, we present the results of Goldmann et al.'s [1] model performance on the selected dataset generated from our study. In figure 5.9 we can see an output image from testing his model on an image of our dataset without training it. The red boxes indicate the predicted ones and the green box indicates the bounding box annotation generated from our app for this image. We can see that most instances of the products on the shelves are correctly localized and this applies for the rest of the results of the testing phase. Additionally, we trained Goldmann's model with our dataset using transfer learning for training on his model. In figure 5.10 one can see an output image as a result of testing our trained model. As with the rest of the testing results, 1-3 bounding boxes were predicted per image.

#### Object detection

In this section, we focus on the results of Lin et al.'s [3] performance on our dataset. For training their model we used transfer learning on a pre-trained model with the dataset provided by Lin et al. [35]. The results of testing the trained model with our dataset were similar to the ones we saw in figure 5.9 with 1-3 bounding boxes predicted per image. In figure 5.11 we can see an output image of testing the model trained with our manually annotated dataset consisting of 28 images with a total of 655 bounding box annotations. The improvement of the model's performance in object localization is noticeable since most of the instances of the products were detected.

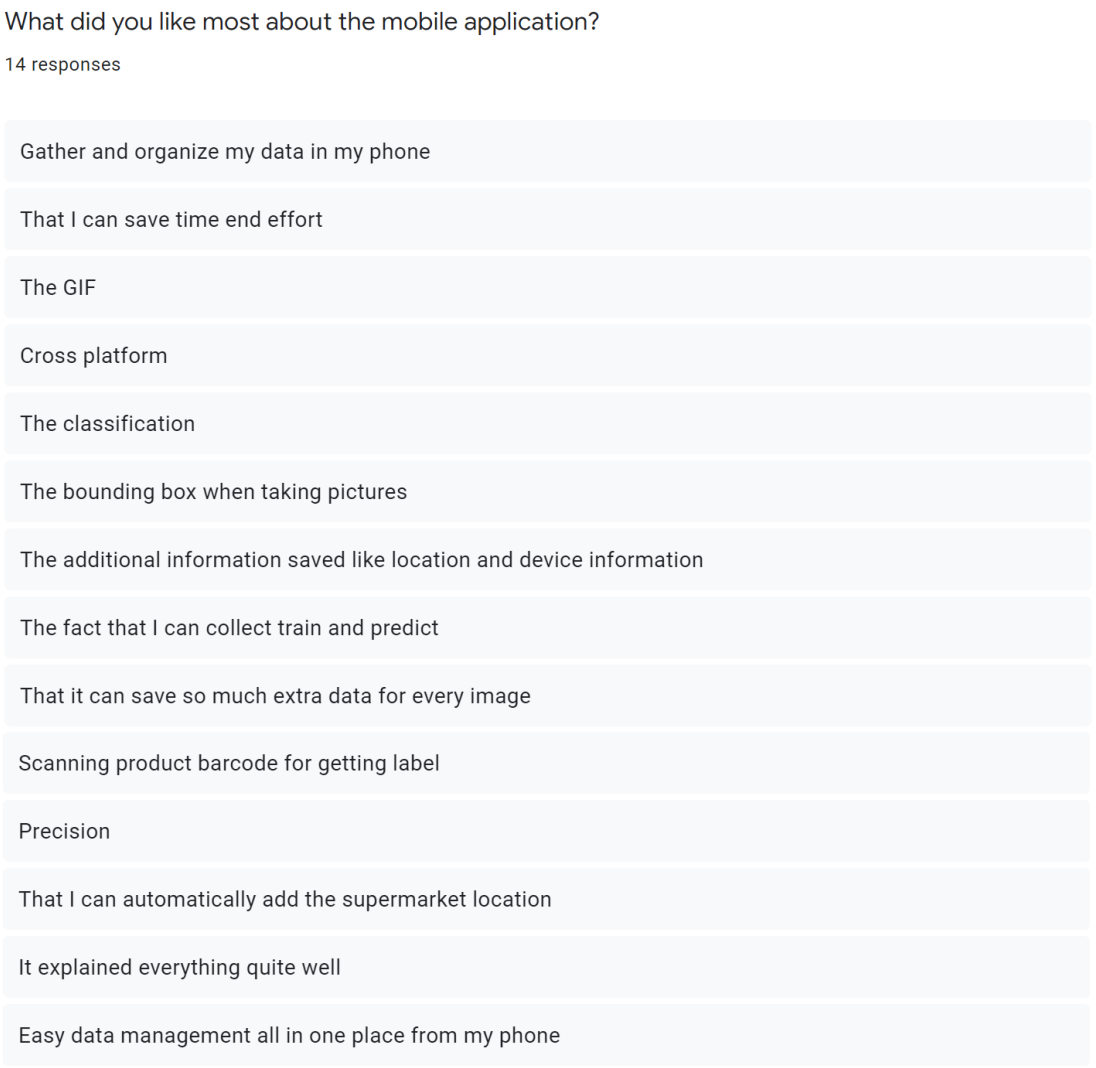



**Figure 5.5**: The results of the third question

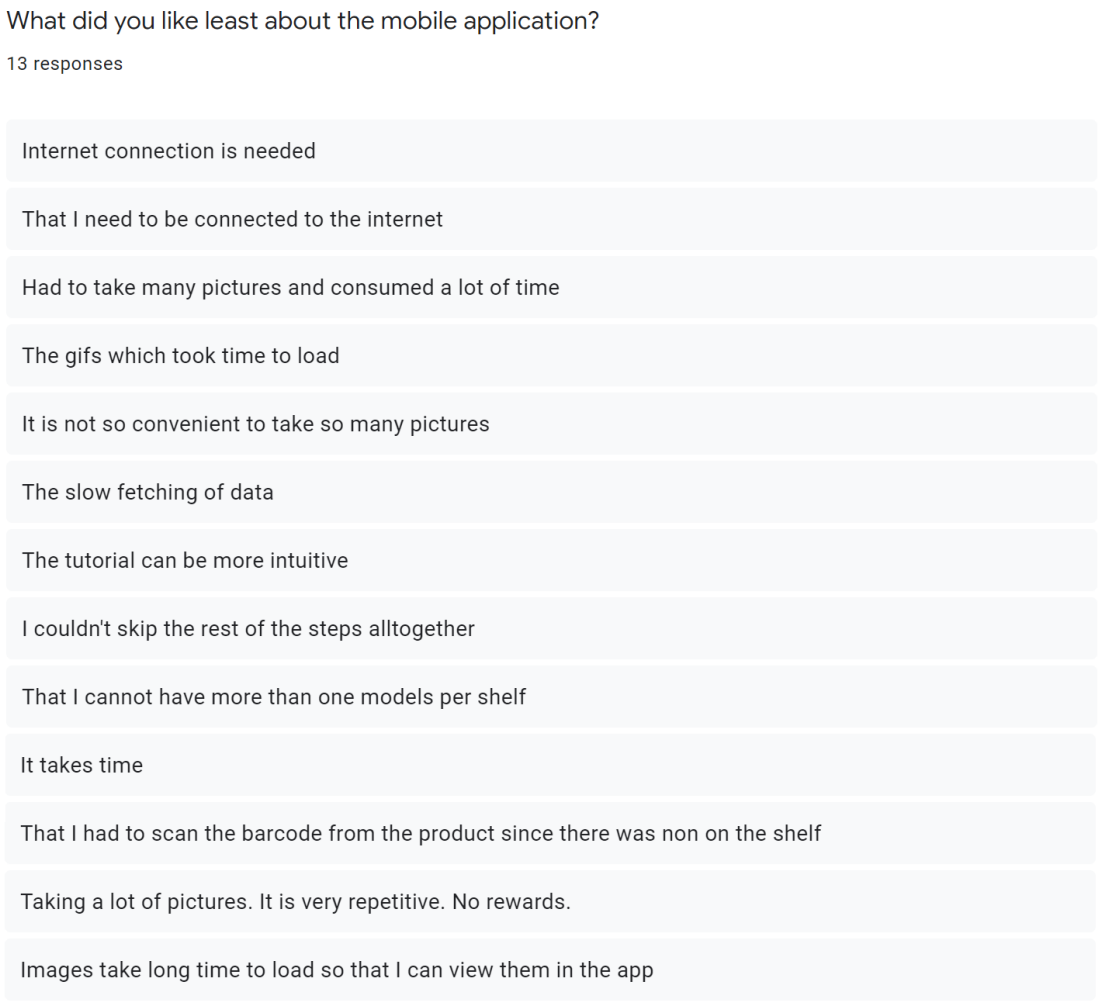



**Figure 5.6**: The results of the fourth question

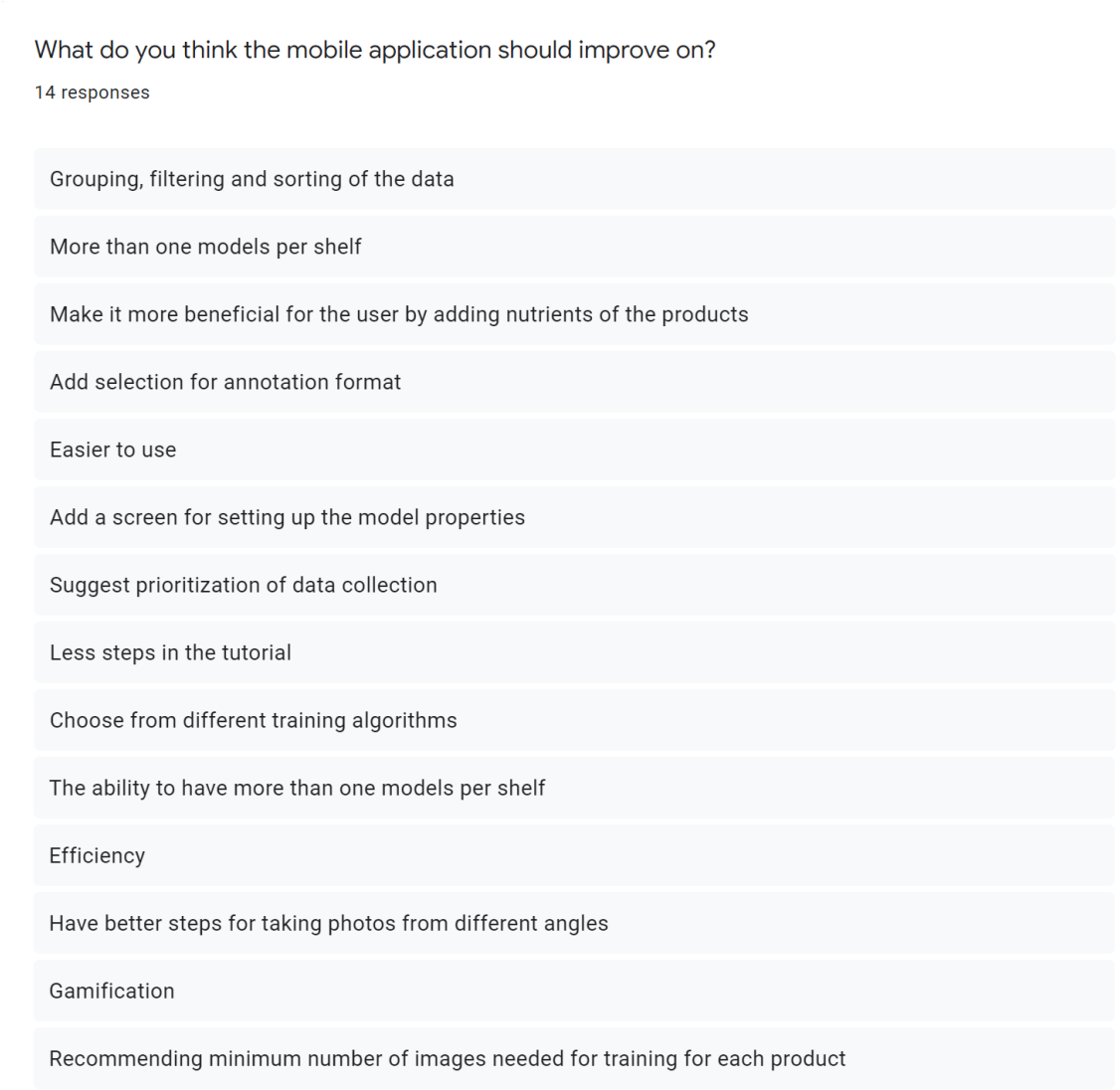



**Figure 5.7**: The results of the fifth question

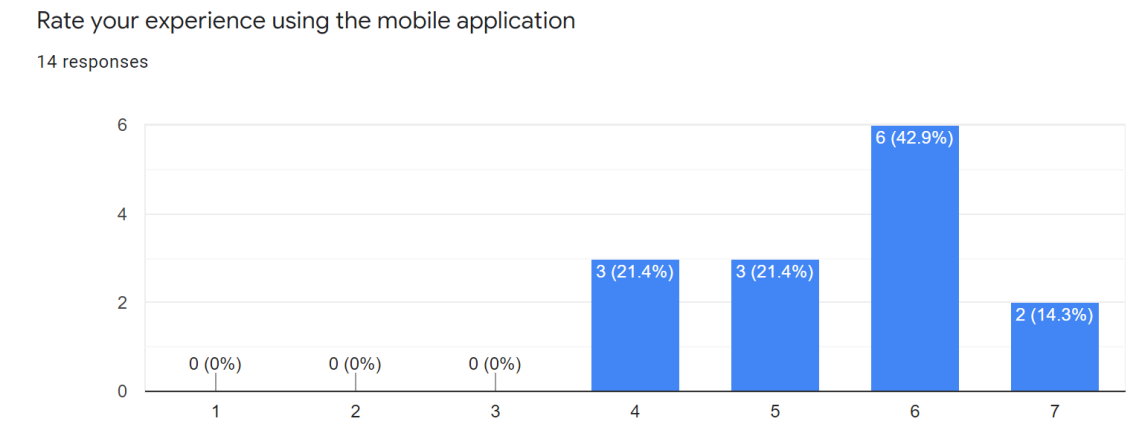



**Figure 5.8**: The results of the sixth question

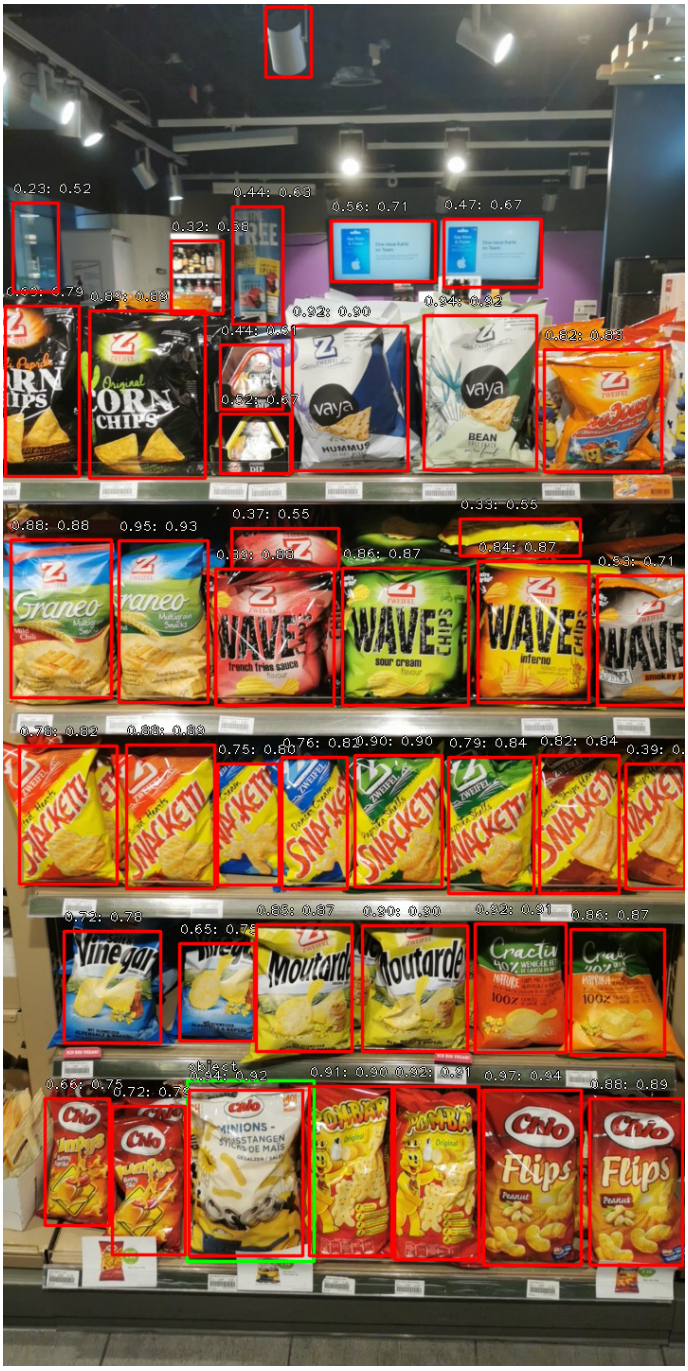



**Figure 5.9**: Result image of testing the object localization model provided by Goldmann et al. [1] without training it with any of our data

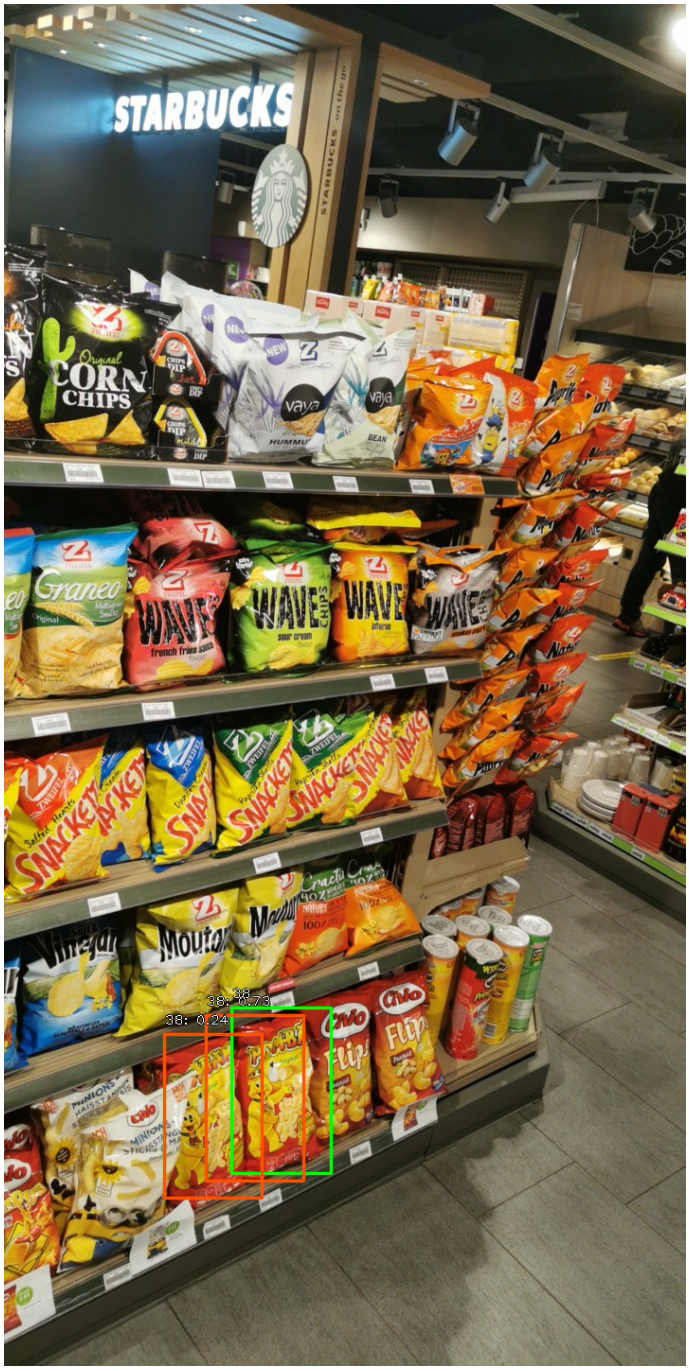



**Figure 5.10**: Result image of testing our object localization model trained with our data using transfer learning from Goldmann et al.'s [1] model


**Figure 5.11**: Result image of testing our model trained with our data using transfer learning from Lin et al.'s [3] model

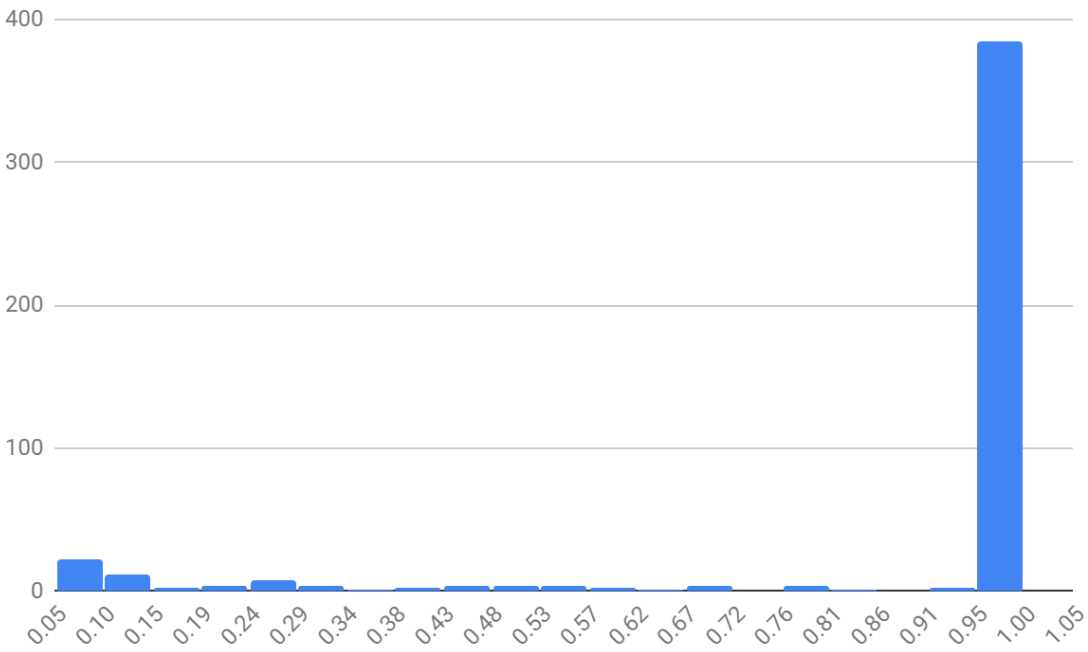



**Figure 5.12**: Histogram of the classification accuracy of 460 predicted bounding box instances of Lin et al.'s [3] model trained on our manually annotated dataset

# Chapter 6

# Discussion

Our usability study shows that the users overall enjoyed using our mobile application and find it a valuable tool. As we can see in the histograms 5.3, 5.4, and 5.8 the results are positive with all answers chosen being above scale four and most of them being on scale six out of seven. Specifically, in the answers of the question regarding the most liked feature of the app, (see figure 5.5) we can see what was most favorable for each user.

Our software system gives the ability to users to collect, organize datasets, and use them for training their deep neural network models as well as predict new unseen data via using their pre-trained models. The fact that all these features are gathered and can be managed all together in the user's phone was mentioned by two of our participants as their most liked feature.

Furthermore, three out of the fourteen answers were related to the additional data, such as location and device information, which is saved together with every photo taken from the user. Other noticeable answers regarding the most liked features of our app include the fact that the app is cross-platform, the convenience in using the bounding box, as well as the scanning of the product barcode for getting the product label.

Most of the answers regarding the least liked features of our app as well as what users think can be improved, which can both be seen in figures 5.6 and 5.7 respectively, do not constitute major problems and most of them can be easily solved. Specifically, two of the answers were about the fact that an internet connection is needed in order to use the application. This indeed can be an issue since in many supermarket stores the phone's signal can be really low or even sometimes it is unreachable. However, this can be easily solved by saving all the images on the phone's local storage and uploading them to our application server only when there is an internet connection. After the local temporary data is uploaded it could safely be deleted from the local phone's storage in order to free up space from the device.

In addition, three answers were related to the slow loading of the statically served images. This issue might be also related to the lack of internet connection which obviously slows the process of fetching statically saved images. The focus of the app is to ease and reinforce the useful generation of labelled datasets. This is why there was not so much emphasis on the performance of the statically served files. The images are locally saved in the machine where the development application server is running. The increase of the speed in which the data is loaded can either be solved by having a separate server just for serving the static files (e.g. Nginx) or by using a cloud-based storage solution.

An additional least liked feature was the fact that the user had to grab the product, find where the barcode is on it, and scan it. Scanning the barcode from the product's package is required because some supermarket stores either do not include a barcode on the product shelf or they use their own barcodes which are different from the ones included on the package. Nonetheless, if there is no barcode on the shelf the user has two options for adding the label of the product. One

can either manually type the product name in our text input or scan the barcode on its package. Scanning should be preferred since the barcode is going to be saved together with the image and this label will be able to be fetched from our database in the future barcode scans.

Another easily fixed feature included in the answers of both 5.6 and 5.7 is the fact that the user cannot have more than one model per shelf. At the moment there is only one model associated with a dataset. Still, our application can use multiple models just by implementing utility functions as a compatibility layer between the deep neural network architecture and our application server's database data format. By adding a list popover, including a list of the available models, when one either presses the train or classify buttons in our mobile application he or she can choose which models to use.

Likewise, the answer in figure 5.7 regarding the selection of annotation format can be solved in a similar way by adding a popover with a list of desired annotation formats the user wants to generate when training his model. Yet, this could be redundant since a selection for a model that he wants to train assumes that the annotation format is going to be converted so that it is compatible with that model. Even so, this feature could be added as an export functionality in case the user wanted just to generate the training data in his desired format without triggering a model training.

The grouping, filtering, and sorting of the data were also mentioned in the answers to improvements and are trivial functionalities nowadays which are easily implemented. Interestingly, the abundant metadata saved from our application combined with a filter functionality can create groups of data that could add great value to our software system. For example, if we group the datasets by supermarket location we can then train a model only with selected images taken from a specific supermarket store. We can then use this model in the future whenever a user wants to perform inference in that store. This would result in better performance since the inference image would be taken in the same environment as the one in which the input image was taken which is used for training. The same would also apply for grouping by other information included in the metadata such as the device model.

Additional useful feedback on how we can improve our application was having the ability to setup the model properties. Currently, all the model training, validating, and testing parameters are included in the command which is used to trigger those actions. Instead, we could ask the user to specify his desired set up by choosing them as inputs in the mobile application and using them instead of the hard-coded ones.

Two of the answers in figure 5.7 were about the suggestion of prioritization of data collection as well as the minimum number of images needed for training each product. The app could only allow training a model if the dataset quantity is favourable. This could be solved by defining a minimum number of images per product, removing low-frequency labels, or having at most a predefined number of more images for the most common product than the least common product. Furthermore, when collecting data the app can recommend working on datasets that would need fewer data in order to be ready for training.

As for the dataset evaluation, if we compare Goldmann et al.'s [1] model result in figure 5.9 with the result in figure 5.10 of our model we see that our generated dataset does not locate all the objects in the image but rather, it detects one to three bounding boxes per image. With this in mind, our assumption was that this was caused because only one bounding box is included per image in our generated data. This assumption is backed up by previous research work which found that training images with missing annotations of objects have a radical impact on the object detection performance [4]. Figure 6.1 demonstrates the negative effect of missing annotations on the performance of object detectors. Therefore, we used the manually annotated dataset of our app including bounding boxes for all the instances of objects of interest in our image. Consequently, as we can also see in figure 5.11 the results show that the model's performance improved drastically with localizing most of the instances of the products on the shelves. If we take into

consideration the small size of our dataset of only 28 images we can assume that the performance will be event better for bigger datasets. As a result, we see that missing annotations have a negative impact on the model's performance rendering our application's generated data rather weak for performing object localization. Still, as one can see in figure 5.12 containing the histogram of the classification's accuracy of 460 predicted bounding box instances, our generated datasets are really valuable for performing image classification.

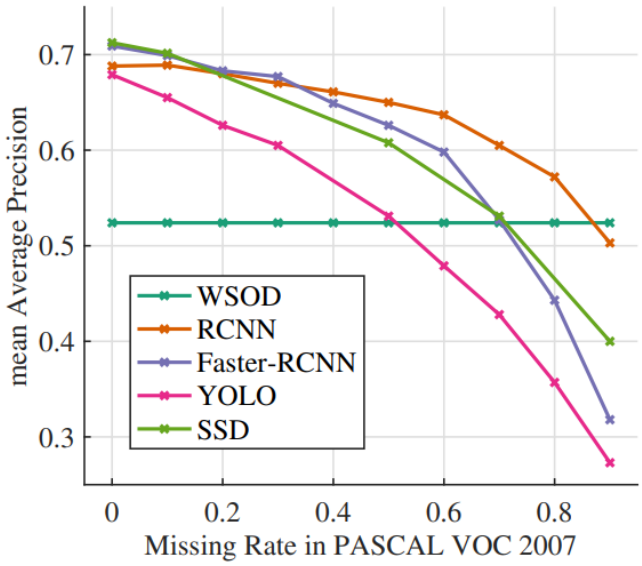



**Figure 6.1**: The Mean Average Precision (mAP) of five detectors training under different instance-level missing label rates of the training dataset. The amount of missing annotations has a significant impact on the performance of the detectors [4].

The user study was conducted in order to objectively determine the value of our software system. However, the following factors may limit the validity of the results:

- **Sample size:**

  The study was conducted with 14 participants being a small number in order to prove statistical significance. Nevertheless, their overall positive feedback is positive.

- **Sample profile:**

  Some of our study participants hadn't used a deep neural network model before and were explained how the manual process of annotating and generating datasets works. Despite our clarification, participants with profound knowledge and experience on the field could give more insightful feedback.

- **The number of deep neural network architectures used:**

  Due to time constraints, we used only two different CNN architectures for evaluating the generated datasets, one for object localization and another for object detection. However, testing our application's generated datasets with different deep neural network architectures can aid in better generalizing our results.

# Chapter 7

# Conclusion

This thesis focused on examining how a mobile application can be developed to make the conventional, time-consuming, manual process of generating and managing meaningful, labelled datasets easier. With this intention, a software system, an overview of which can be seen in figure 3.4, was developed which consists of a mobile client backed by an application server. In order to investigate our approach for developing our software system, I analysed my research design can be seen in Chapter 3.

The users can generate and manage labelled image datasets including useful metadata by using their phone's camera and persist these datasets in a database with the help of the application server. Besides, training, validation, testing, as well as inference on new unseen images, can be triggered by the mobile application by using our computer vision components. These components are included in a modular way in our software system meaning that they can easily be replaced by different deep neural network architectures. Chapter 4 thoroughly describes our methodology and details of the implementation of our software system's components.

A usability study was conducted in order to answer our research questions from Chapter 1:

> **RQ1: Can a mobile application reinforce a faster and more meaningful generation of labelled Convolutional Neural Network datasets?**

The developed mobile application indeed reinforces faster generation of labelled datasets. First of all, the fact that it is cross-platform makes it easier to collaborate with other people, regardless of their phone's operating system, resulting in rapid crowd-sourcing of images all in one place.

The fact that the datasets generated from our application have labels included, which can be fetched by scanning the product's barcode, saves time since the user does not have to manually find and enter the label.

Moreover, every image of our generated dataset includes valuable metadata such as supermarket store location, the angle from which the photo was taken as well as device model information. These metadata can be used in order to filter or group datasets and train models only with the grouped data, e.g. having for every store a separate model trained with dataset images from that particular store.

As we already saw in Chapter 5, the overall positive feedback received from our usability study further supports the value of our mobile application and as a result of our software system in general.

> **RQ2: Is a mobile application sufficient for replacing the manual creation and annotation of datasets for both object localization and object classification while having similar accuracy?**

The answer to this question is partial yes. As we saw in figure 5.12 the classification results render our application a sufficient replacement for the image classification task. On the contrary, the figure 5.10 designates the weakness of our application caused by missing bounding box annotations which negatively impact the performance of the model. Given that, our application is not a good candidate for replacing the manual object localization task.

In closing, I can state that our mobile application already adds great value and eases the tedious process of generating labelled datasets ready to be used for training, especially for image classification. I strongly believe that it has the potential of becoming a full substitute for the conventional way of manually gathering and labelling training data as well as making predictions not only for the image classification but also for the object detection task in general.

The tutorial that we used on the camera screen could be more entertaining for the user by adding gamification features. As we can see in both figures 5.6 and 5.7 eight out of twenty-seven answers were about the fact that our six-step tutorial took a lot of time and that the users would prefer having some rewards while taking the photos. Researching valuable gamification features would further reinforce the users to collect training data in a more enjoyable way.

The poor performance of the object localization model trained with our dataset due to only one bounding box created the need to be able to label all instances of objects in an image in the mobile app. Provided that we saw promising results using our manually annotated datasets, one could research an intuitive way of adding bounding boxes in the app around all the objects of interest in an image. This could be done by taking the photo first and then adding the bounding boxes in contrast to the current implementation in which the bounding box is added before we take the photos on the camera screen.

Last but not least, the datasets generated from our software system should be evaluated with various deep neural network architectures of object localization, image classification, and object detection.

# Bibliography


[1] E. Goldman, R. Herzig, A. Eisenschtat, J. Goldberger, and T. Hassner, "Precise detection in densely packed scenes," in *Proc. Conf. Comput. Vision Pattern Recognition (CVPR)*, 2019.

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available:

[3] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[4] A. E. Abdel Hakim and W. Deabes, "Can people really do nothing? handling annotation gaps in adl sensor data," *Algorithms*, vol. 12, no. 10, p. 217, Oct 2019. [Online]. Available:

[5] W. Geng, F. Han, J. Lin, L. Zhu, J. Bai, S. Wang, L. He, Q. Xiao, and Z. Lai, "Fine-grained grocery product recognition by one-shot learning," 10 2018, pp. 1706–1714.

[6] L. Karlinsky, J. Shtok, Y. Tzur, and A. Tzadok, "Fine-grained recognition of thousands of object categories with single-example training," 07 2017, pp. 965–974.

[7] A. Tonioni, E. Serra, and L. Di Stefano, "A deep learning pipeline for product recognition on store shelves," 12 2018, pp. 25–31.

[8] A. Tonioni and L. Di Stefano, "Domain invariant hierarchical embedding for grocery products recognition," *Computer Vision and Image Understanding*, 03 2019.

[9] K. Fuchs, T. Grundmann, and E. Fleisch, "Towards identification of packaged products via computer vision: Convolutional neural networks for object detection and image classification in retail environments," 10 2019, pp. 1–8.

[10] M. George, D. Mircic, G. Soros, C. Floerkemeier, and F. Mattern, "Fine-grained product class recognition for assisted shopping," 12 2015, pp. 546–554.

[11] A. Franco, D. Maltoni, and S. Papi, "Grocery product detection and recognition," *Expert Systems with Applications*, vol. 81, 03 2017.

[12] P. Pouladzadeh, A. Yassine, and S. Shirmohammadi, "Foodd: Food detection dataset for calorie measurement using food images," vol. 9281, 09 2015.

[13] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" 01 2014, pp. 3320–3328.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *IEEE International Conference on Computer Vision (ICCV 2015)*, vol. 1502, 02 2015.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 09 2014.

[16] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," 06 2015.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[18] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," 06 2016, pp. 1–6.

[19] S. Belongie, M. Merler, and C. Galleguillos, "Recognizing groceries in situ using in vitro training data," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2007, pp. 1–8. [Online]. Available:

[20] T. Winlock, E. Christiansen, and S. Belongie, "Toward real-time grocery detection for the visually impaired," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, 2010, pp. 49–56.

[21] L. Barrington, T. K. Marks, J. H.-w. Hsiao, and G. W. Cottrell, "Nimble: a kernel density model of saccade-based visual memory," *Journal of vision*, vol. 8, no. 14, p. 17.1—14, November 2008. [Online]. Available:

[22] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[23] A. Tonioni, "Computer vision and deep learning for retail store management," Ph.D. dissertation, alma, Aprile 2019. [Online]. Available:

[24] M. George and C. Floerkemeier, "Recognizing products: A per-exemplar multi-label image classification approach," in *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 440–455. [Online]. Available:

[25] A. Mukhija, "Image2product: A deep learning pipeline for product recognition in densely packed scenes," Master's thesis, ETH Zurich, 05 2020.

[26] R. Girshick, "Fast r-cnn," 2015.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[29] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," 2014.

[30] OpenFoodFacts contributors, "Open food facts the free food products database," 2020, [Online; accessed 08-August-2020]. [Online]. Available:

[31] Auto-ID Labs team, "The foodcoach project." 2020, [Online; accessed 12-August-2020]. [Online]. Available:

[32] G. Jansen, "Thoughts on restful api design. lessons learnt from designing the red hat enterprise virtualization api," , 11 2012, accessed: 2020-08-18.

[33] J. B. K. Q. Weinberger and L. K. Saul., "Distance metric learning for large margin nearest neighbor classification." 2006.

[34] ngrok contributors, "Introspected tunnels to localhost," 2020, [Online; accessed 22-August-2020]. [Online]. Available:

[35] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2014.

# Appendix

The following links contain the source code of our mobile client and our back-end server. Both include documentation on how to be installed as well as used. The application server and the two computer vision components are added as git sub-modules in the attached server repository.

- **Mobile application source code:** https://github.com/MariosVisos/retail-product-classifier
- **Server source code:** https://github.com/MariosVisos/rpcserver