Chair of Digital Image Processing,
Faculty of Computer Science and Mathematics
University of Passau 2023

Master's Thesis

Computer Vision for determination of fridge contents

Mohamed Bokhari

1. Examiner	2. Examiner
Prof. Dr. Tomas Sauer	Prof. Dr. Michael Granitzer

Author

Bokhari Mohamed
Chair of Digital Image Processing
University of Passau

Examiners

Prof. Dr. Tomas Sauer, University of Passau
Prof. Dr. Michael Granitzer, University of Passau

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel `Computer Vision for determination of fridge contents` selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe und dass alle Ausführungen, die wörtlich oder sinngemäss übernommen wurden, als solche gekennzeichnet sind.

Mit der aktuell geltenden Fassung der Satzung der Universität Passau zur Sicherung guter wissenschaftlicher Praxis und für den Umgang mit wissenschaftlichem Fehlverhalten vom 31. Juli 2008 (vABlUP Seite 283) bin ich vertraut.

Ich erkläre mich einverstanden mit einer Überprüfung der Arbeit unter Zuhilfenahme von Dienstleistungen Dritter (z.B. Anti-Plagiatssoftware) zur Gewährleistung der einwandfreien Kennzeichnung übernommener Ausführungen ohne Verletzung geistigen Eigentums an einem von anderen geschaffenen urheberrechtlich geschützten Werk oder von anderen stammenden wesentlichen wissenschaftlichen Erkenntnissen, Hypothesen, Lehren oder Forschungsansätzen.

[illegible]

....

(Bokhari, Mohamed)

Abstract

People are now more willing to improve their lifestyle by taking care of their health and knowing about their eating behaviour and habits as technology advances and improves. With the advancement of neural networks and image processing techniques, we can now monitor our food preferences by continuously tracking the contents of our refrigerator, allowing us to track our eating behaviour. Since smart refrigerators from international companies like LG, Samsung and Bosch are so expensive, we decided to investigate how we can build a model into a refrigerator that can initially detect target objects. Then add more functions in the future, such as suggesting a shopping list based on the detected objects, and this could help in the future to improve the market and lower prices.

In this thesis, we investigated the factors through several experiments that could aid in the discovery of the optimal way to detect the objects (food) stored in our refrigerator. We focused our research on fine-tuning and training a custom object detector Faster R-CNN using the Tensorflow object detection API to detect four objects. Then in future works, it would be simple to expand the number of objects based on the discovered factors.

In this work, we used the Faster R-CNN model pre-trained on COCO17 and conducted experiments on two refrigerators. We tried several methods and conditions to determine what could help improve model accuracy, such as calibrating the number of the dataset, changing the light condition and taking photos from different angles. Finally, the findings of our study revealed that the combination of these various factors enhanced the model's performance and resulted in the successful detection of the four objects stored in our refrigerator.

Acknowledgements

I would like to acknowledge and give my warmest thanks to my supervisor, Prof Dr. Sauer, who made this work possible. His guidance and advice at the beginning of the thesis helped me a lot to define the scope and focus of the thesis and formulate the research questions. I would also like to give special thanks to my second supervisor, Prof. Dr. Granitzer, for taking the time to supervise my thesis. I would like to thank my father and my sister for their continuous support through all the stages of writing my thesis and for giving me suggestions and feedback to improve it. Thank you so much for your contributions, and I wish you all the best in your life. Finally, I would like to thank my family as a whole for their support and for everything they have done for me until I succeed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Smart refrigerators, such as the Samsung Family Hub and the LG Smart InstaView, are costly and use object detection technology to track the content stored inside them.

In this study, we aim to transform a traditional refrigerator into an intelligent refrigerator by incorporating object recognition capabilities without incurring a significant monetary expense. Our objective is to investigate the factors that enhance the precision of object detection in a refrigeration unit.

We employ a methodology of fine-tuning a pre-trained object detector, Faster R-CNN, utilizing a dataset that we have created and annotated, gathered from two refrigerators under varying conditions and using various techniques. These intelligent refrigerators, however, are generally more costly than traditional refrigerators and typically operate on a proprietary platform that is not readily accessible for research and development.

1.1 Motivation

We frequently get confused about which dish to prepare and waste time thinking about it. The concept of smart homes has now become a reality thanks to the Internet of Things, Machine Learning, computer vision, image processing, and other technological developments.

Innovative refrigeration is another intelligent home component that uses IoT to simplify human life by providing various control options and advanced features.

Because of advances in computer technology and the widespread use of the Internet, intelligent systems with multimedia and innovative operation capabilities have begun to appear in our daily lives. One of the places where we could find such intelligent systems is in the kitchen.

Because modern life gives people less time due to their busy schedules, they do not have much time to think about which recipe to cook, what items they have in their fridge, what else they need to buy, and so on. Hence, an intelligent system, such as a smart fridge, can help us live a more enjoyable and healthy lifestyle.

We commonly have features that help track inventory, set the user preferences for recipes, and get recipe recommendations - to create an intelligent fridge with the additional functionality of helping the person save time by not wasting time thinking about which recipe to make today.

As time has become more important in the busy life of a modern person on a global scale, this additional feature could help increase the commercial appeal of an Intelligent Fridge. We believe an Intelligent Fridge will play a significant role in future smart homes.

1.2 Objectives

In this paper, we present a new approach for identifying bell pepper, tomato, carrot, and butter on the top shelf of a refrigerator. To achieve this goal, we conducted experiments using two refrigerators by collecting datasets from both and comparing the results to determine the factors that could help improve the model's accuracy. We created approximately 1000 images of two refrigerators for the four classes and annotated them. In the training data, we isolated each class and took photographs of them. In the test data, we combined the four classes in different ways and positions to ensure the independence of the training data from the test data. The object detector's performance was then evaluated using various data calibrations, adjustments, and factors. We then adopted the ones that provided minimal errors and demonstrated these optimal choices, resulting in improved results.

1.3 Scope

This thesis aims to develop an evaluation method for visual datasets by studying and analyzing two diverse datasets from two different refrigerators used in developing intelligent fridge detection systems. The TensorFlow Object Detection API is used to develop, train and evaluate these systems. In addition, we compared and analyzed the datasets used in the training process to determine essential features that impact generalized food detection ability and accuracy. We used Tensorflow and the Tensorflow object detection API to train our custom object detector. Specifically, we took photos of two fridges from two different homes to collect two different datasets, used the LabelImg tool to label the images manually, trained the model on Denethor, an Ubuntu machine equipped with an NVIDIA Corporation GV100 [TITAN V] graphics card located in the FORWISS sub-network provided by the University of Passau, and used Tensorboard to visualize the results.

1.4 Definition of terms

1.4.1 Machine Learning

It is a subset of the artificial intelligence area. The science of learning from experience and improving the performance as it learns. It is a discipline that helps in optimizing processes and resource allocation. The word "learning" indicates improvements could be possible in the future.

1.4.2 Neural Networks (NNs)

Are a machine learning subfield that forms the foundation of deep learning algorithms. The human brain inspires its name and structure and mimics how biological neurons communicate. They allow computer programs to recognize patterns and solve common problems in AI, machine learning, and deep learning.

1.4.3 Deep Learning (DL)

Is a machine learning subfield that refers to training large neural networks. These neural networks attempt to mimic human brain behaviour by "learning" from large amounts of data. It allows systems to gather data and make amazingly accurate predictions.

1.4.4 Computer Vision

Focuses on the challenge of developing intelligent visual systems capable of understanding and interpreting our real, three-dimensional world by allowing computers and systems to derive meaningful information from digital images, videos, and other visual inputs.

1.4.5 Object Detection

It is a computer vision technique that uses a bounding box with a label to locate and recognize objects in images or videos. It could count objects in a scene, determine and track their precise locations, and label them accurately.

1.5 Outline

We divided this thesis into three sections: an introduction, five main chapters listed below, and a conclusion.

1.5.1 Chapter 2: Background

In this section, we thoroughly describe this thesis's key concepts and theoretical foundations. We begin by presenting the fundamentals of machine learning, artificial neural networks, and deep learning. Then, we discussed the characteristics of convolutional neural networks and described the Faster R-CNN model and its predecessors. Furthermore, we demonstrate the ability to utilize pre-existing models and algorithms by applying transfer learning techniques. This includes the utilization of well-known CNN architectures and the TensorFlow Object Detection API to fine-tune a pre-trained object detector model that has already been trained on a large-scale dataset.

1.5.2 Chapter 3: Related Work

This section reviews the literature on the current state-of-the-art techniques in object detection methods in refrigerators and related fields. The review covers the most recent developments and advancements in the field, providing an overview of the current state of the art and identifying potential areas for further research.

1.5.3 Chapter 4: Implementation

In this section, we outlined the methodology for developing and assessing the object detection system, including the evaluation metrics employed.

1.5.4 Chapter 5: Results

In this chapter, we evaluated and analyzed the performance of the fine-tuned model on the test dataset. The results of evaluating our object detector are presented, specifically focusing on the performance of fine-tuning the model on the two created datasets to identify the factors that lead to improving the model's accuracy and performance for our application.

1.5.5 Chapter 6: Discussion

In this chapter, we summarized our results and discussed the factors that helped improve the accuracy of our model. We also discussed some limitations of our work and offered insights and recommendations for future research.

Chapter 2

Background

The success of AlexNet in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 won by Krizhevsky et al.[10], has dramatically impacted the field of computer vision, particularly in the domain of object detection.

Object detection in a refrigerator is a crucial task in the field of computer vision, particularly in the areas of food management and home automation. The objective is to investigate the factors affecting object detection accuracy to accurately detect and classify objects inside a refrigerator, such as food items, drinks, and other household items. This is a challenging task due to the complex and dynamic nature of the refrigerator environment and the wide variety of objects that can be present. The potential applications of this research include improving food management and reducing food waste, as well as providing assistance to individuals with disabilities or limited mobility. The research will likely involve using techniques such as deep learning, convolutional neural networks, and other computer vision algorithms to analyze images and extract features, as well as developing new methods specifically tailored for object detection and classification in a refrigerator environment. This work will also involve studying various challenges specific to the refrigerator environment, such as low light, occlusion, and variations in object size and shape.

2.1 Theory

This section describes some fundamental theoretical aspects related to this thesis. This section aims to provide important information concerning the theory behind food/content in fridge detection.

2.1.1 Machine Learning

Machine learning is the science of learning from experience to predict the future. It involves many experience-based predictions and improves system performance by learning from experience using computational methods [22].

We can translate this by considering the experiences in computer systems as data. The goal is to build models from the data by developing learning algorithms, which could be achieved by feeding the learning algorithm with experience data to obtain a model that can make predictions [22].

So, to perform machine learning, we first need to collect a dataset and then use the learning concept to build models using learning algorithms; the data used in the training phase is called training data.

The evaluation phase of our learned model is called testing. We use a test set to evaluate the model, called testing data, to see how well the model has learned to approximate the ground truth by learning from the training set.

When we try to predict a discrete output, we call this a classification problem, and if the prediction output is continuous, it is called a regression problem.

We distinguish between binary and multiclassification. In binary classification, there are two types of classes, one positive and one negative; if there are more than two classes, it is a multi-class problem.

2.1.2 Artificial Neural Network (ANN)

The human brain can swiftly recognise objects from various perspectives and under varying lighting conditions, even in a cluttered environment. However, in the past, computer systems were not able to perform such tasks with a similar level of efficiency.[11] Neural networks form the basis of deep learning, a machine learning subfield where the structure of the human brain inspires the algorithms. Neural networks absorb data, train themselves to acknowledge the patterns in this data, and then predict the outputs for a brand-new set of comparable data. Layers of neurons make up neural networks. These neurons are the core processing units of the network. Artificial neural networks (ANN) consist primarily of layers: an input layer, an output layer, and one or more hidden

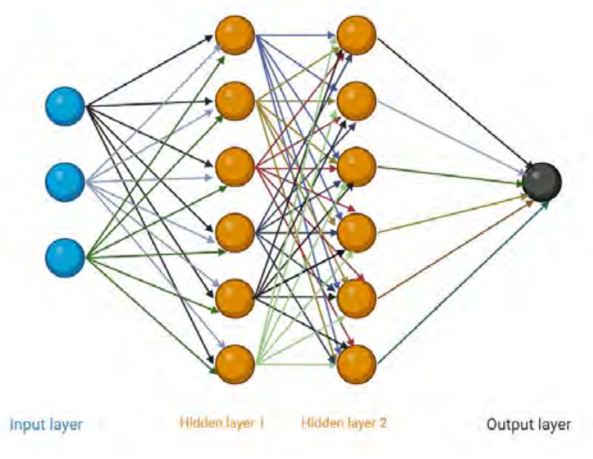


Figure 2.1.1: Artificial Neural Network Architecture

layers. The first hidden layer learns simple and basic features and passes them to the second hidden layer.[1] Supported by the inputs from the previous layer, the second layer learns more complex and abstract features than the primary one.[1]

It is likely that deeper layers within the network possess the capability to learn features that are more intricate and advanced in comparison to the ones in the upper layers. A multi-layered network can effectively address complex issues such as image classification and object detection through this mechanism. [11]

2.1.3 Deep learning

Deep learning constitutes a sub-discipline of machine learning and artificial intelligence, and it pertains to the scientific study of utilizing multi-layered artificial neural networks to train computational models. These neural networks, called "deep neural networks," are characterized by multiple hidden layers between the input and output. The term "deep" in "deep learning" denotes the depth of the neural networks, which typically have more than two layers beyond the input and output layers. These deep networks have demonstrated remarkable accuracy improvements across various applications, including image classification, object recognition, speech recognition, and language translation. The primary approach employed in deep learning is learning representations (or features) from data, such as text, images, or videos, rather than implementing task-specific algorithms. This learning process can be either unsupervised or supervised, with the latter being the most prevalent in practical systems. One of the critical characteristics of deep learning is that an increase in the amount of training data leads to an improvement in performance, unlike traditional learning methods that tend to plateau performance

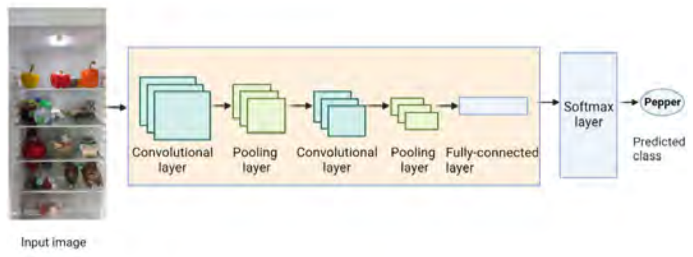


Figure 2.1.2: Concept of the Convolutional Neural Network Architecture

beyond a certain data threshold. Additionally, deep learning models are known for their hierarchical feature learning capabilities. Lower-level features are learned in the early layers and are subsequently passed on to the higher layers to learn more complex features. The scalability and hierarchical feature learning capabilities of deep neural networks render them particularly appropriate for tasks like image classification, object recognition, and speech recognition.

2.1.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific category of artificial neural networks that are based on convolution operation. This operation utilizes a window or kernel, referred to as a filter, which is moved across the input image to learn and identify features within the image. The underlying principle behind this operation is that it is more efficacious to examine smaller portions of an image to detect specific features rather than analyzing the entire image at once. CNNs are composed of multiple convolutional layers, which distinguishes them from other types of artificial neural networks. They are extensively utilized in visual recognition tasks, such as image classification and object recognition. Due to its exceptional performance in the ImageNet Challenge in 2012 [10], CNNs have gained widespread popularity and usage among computer vision researchers and in the broader computer vision community.[1]

The figure above shows a typical basic architecture of CNN, which are:

1. Convolution layers
2. Pooling layers
3. Fully connected layers
4. Softmax function

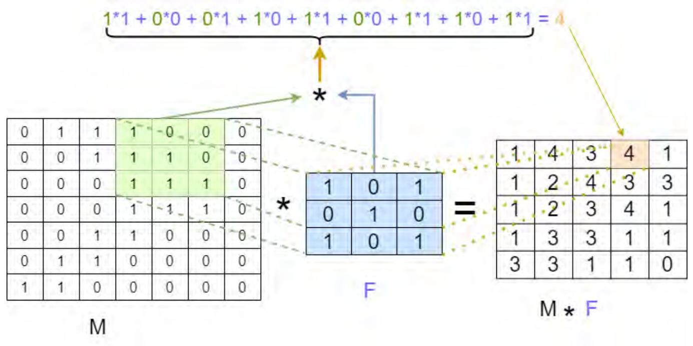


Figure 2.1.3: An example of convolution operation

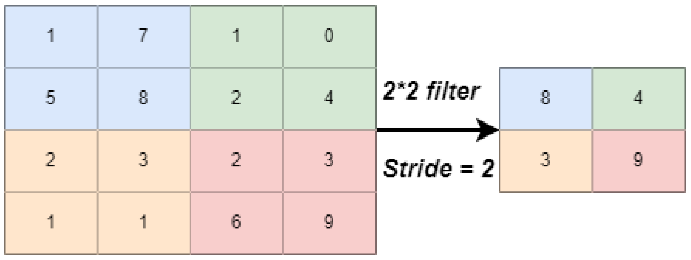


Figure 2.1.4: Max Pooling example with 2*2 filter and stride of 2

The input is an image that goes through several convolutional layers and pooling layers, followed by some fully connected layers. In the end, the softmax function is applied to perform a multiclass classification.

Convolution layers:

The convolution layer constitutes the fundamental and crucial building block of a CNN. These layers are composed of neurons that function as filters. The filters are systematically traversed across the entire image to generate feature maps. This traversal process is called convolution (as depicted in Figure 2.1.3). As illustrated in the aforementioned example, the initial block comprises a series of convolution layers. Each layer operates as a filter and learns distinct feature maps from an input image or feature map. For instance, given an image as input, if there are four convolution layers in a block, the output of each layer would be a feature map representing three distinct characteristics, such as color, edges, and shape. Additionally, the output of each convolution block exhibits three dimensions: height, width, and depth. The height and width correspond to the dimensions of the feature maps, while the depth corresponds to the number of convolution layers.[1]

Pooling layers:

Pooling layers generally follow convolutional layers and are used to reduce the size of the representation, speed the computation, and make some of the features detected more robust. There exist various forms of pooling, including but not limited to max-pooling as illustrated in Figure 2.1.4, which is widely utilized, sum-pooling, and average-pooling. The Max-pooling operates in the following manner: a window traverses each feature map, and the maximum value within that window is chosen. We have an input of four by four and want to apply a type of pooling called max-pooling. This implementation of max-pooling with a 2*2 window and a stride of 2 will be a two-by-two output with a stride of 2. In average pooling, a kernel of size n*n is moved over the matrix. For each position, an average of all values is taken and inserted into the initial part of the initial matrix.

Fully Connected Layers (FC):

The fully connected layer is a simple feed-forward neural network and will add up after some convolutional and pooling layers at the end of the neural network. The fully connected layer consumes a one-dimensional input, so the three-dimensional output of the convolutional and pooling layers must be converted to a one-dimensional layer and passed to the fully connected layer. Flattened means that the entire three-dimensional matrix of the pooling and convolution layer is rolled up into a vector.

Softmax function:

After passing through the fully connected layers, the last layer uses the softmax activation function (instead of ReLU) to determine the probabilities of the input belonging to a particular class (classification). The softmax function is more convenient for multi-class detection than the sigmoid function, which is helpful for binary classification. The result of the softmax function for multiple class detection is a vector containing all probabilities for each class, and the class with the highest probability is the target class. The Softmax function formula is as follows:

$$\sigma(a)_i = \frac{e^{a_i}}{\sum_{j=1}^{K} e^{a_j}}$$

$$For: \; i = 1, ..., K$$

Where:

- a: Input vector $(a_0, ..., a_K)$ to be passed to the softmax function.
- a_i: represents all the elements of the vector passed to the softmax function, and they can take any real value.
- K: represents the number of classes in the multi-class classifier.
- e^{a_i}: Each element of the input vector is applied to the standard exponential function. This results in a positive value greater than zero, which is very small if the input is negative and very large if the input is significant. It must, however, be fixed in the range (0, 1) required for probability [20].
- $\sum_{j=1}^{K} e^{a_j}$: This represents the normalization term for ensuring the values of the output vector $\sigma(a)$ will sum to 1 [20].

2.1.5 Important hyperparameters used in CNN:

Padding:

Building deep neural networks causes the input image to shrink with each convolutional operation. So, if we have a hundred layers of a deep network, it shrinks a little bit on each layer until we get a tiny image after a hundred layers, and the second issue is that much information is thrown away at the images edges.

When we look at a pixel at the corner or edge, we see that it is only utilized in one of the outputs since it touches the three-by-three regions that overlap that pixel. As a result, it appears like pixels near the corner or edge are used significantly less in the output, resulting in the loss of much information at the images edge. We may pad the image to address both difficulties. By convention, we padded with zeros, and if p equals the padding amount, since we are padding all around with an extra margin of one pixel, p is equal to one in this example, and the output size becomes $(n + 2p - f + 1) * (n + 2p - f + 1)$

We differentiate between two kinds of convolutions:

- Valid Convolutions: This means that we are using no padding, and as shown in Figure 2.1.5, we are convolving a 6*6 by 3*3 filter, and this would give a 4*4 dimensional output

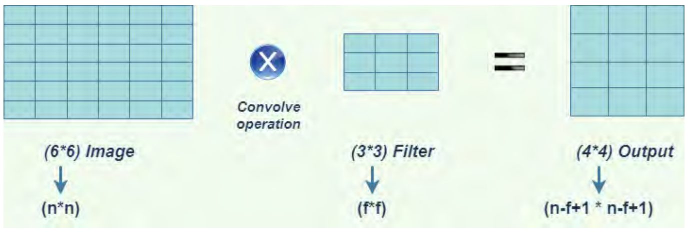


Figure 2.1.5: Valid convolutions

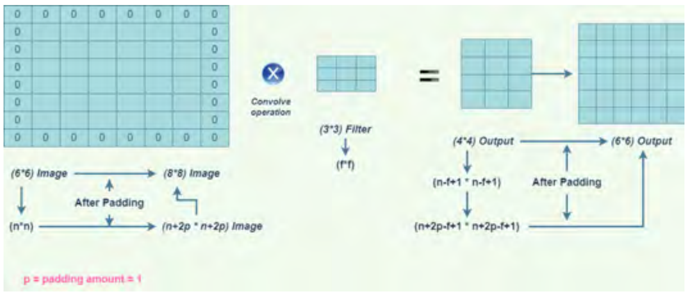


Figure 2.1.6: Same convolutions

- Same Convolutions: We use padding here so that the output dimension is the same as the input size, as shown in Figure 2.1.6

Stride:

The stride is the choice of how many pixels the analysis window moves when scanning the input layer on each iteration when applying convolutional operations. The majority of CNN's use a fixed stride value [21]. Figure 2.1.7 shows an example of a stride with one and a stride with 2. In the stride of 2, each kernel is 2 pixels displaced from its predecessor.

2.1.6 Supervised Learning

Machine learning is detecting hidden patterns in data and then using those patterns to categorize or predict an event related to the problem [2]. "Supervised Learning" is a machine learning paradigm typically employed in deep learning. It is the ability of an algorithm to generalize knowledge from existing data with the target or labelled examples such that the approach may predict new (unlabeled) cases. An ideal case would be for a supervised learning system to accurately select the correct class labels for unseen data. A model is trained using supervised learning on a dataset with known correct answers. When the model fails to predict the answer, the mapping function is adjusted to decrease error loss by modifying weight vectors. When the system achieves a desirable level of

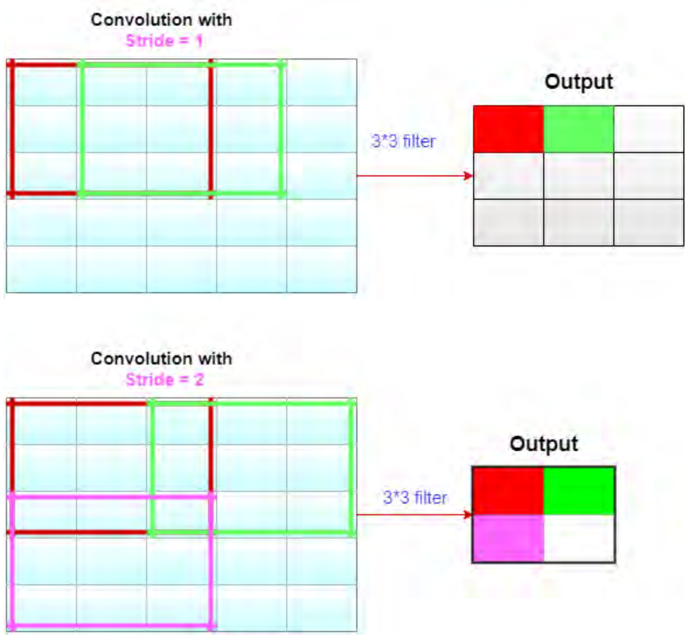


Figure 2.1.7: Stride convolution example when s=1 and s=2

performance, which means it reliably predicts the output of a given input, the training is complete.

2.1.7 Object Detection

Object detection is a computer vision technique that detects objects in images and videos. It combines image classification into a specific label class with object localization within the image (see Figure 2.1.8), making it more challenging and complex than image classification or object localization alone. Object detection is used in many applications and is being massively debated in autonomous vehicles.

2.1.8 Transfer Learning

Training machine and deep learning models can be complex due to the lack of annotated training data in a given domain. It is time-consuming and expensive to train models from scratch and could require a lot of GPUs for object detection tasks. Reducing the need and effort to recollect the training data and build the models from scratch would be nice. In such cases, knowledge transfer or transfer learning between task domains would be an optimal option [15].
Transfer learning is a machine learning technique based on reusing past learned knowledge and applying that knowledge to solve another related problem [15]. This technique can be applied to many machine learning models, including deep learning models. The goal

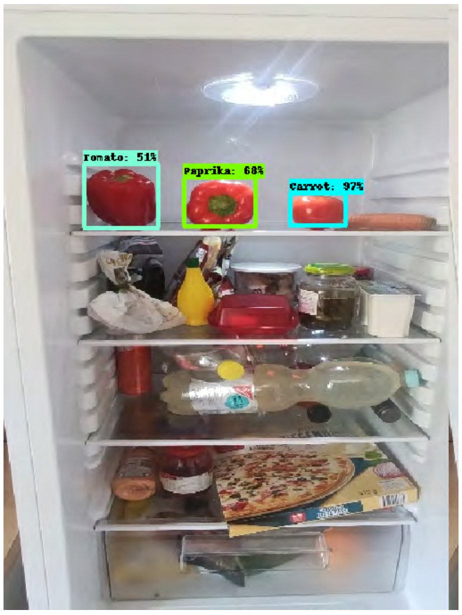


Figure 2.1.8: Object Detection example

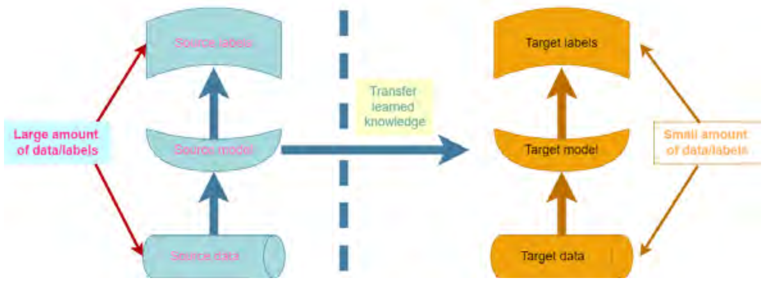

Source labels
Source model
Source data
Large amount of data/labels
Transfer learned knowledge
Target labels
Target model
Target data
Small amount of data/labels

Figure 2.1.9: Concept of transfer learning

is to use the knowledge of a pre-trained model while performing a different related task (Figure 2.1.9).
The benefits of transfer learning are that we can achieve higher accuracy with less training data and train faster because the original model has already been trained on a large dataset. Thus, we do not have to begin training from scratch, which would be a better start for solving a target problem.

Fine-Tuning

Fine tuning is associated with the concept of Transfer learning. It is a method of implementing or utilizing transfer learning. Fine-tuning is a process that takes an already trained model for a given task and tunes it to perform a second similar task, presuming that the initial task is similar to the new one. Using a pre-designed and trained artificial neural network allows us to leverage what the model has already learned rather than develop it from scratch.

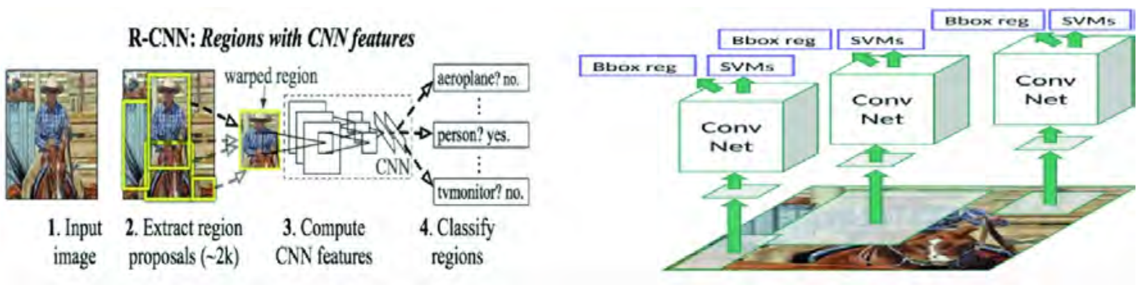


Figure 2.1.10: R-CNN Architecture
[5]

Building models from scratch can be a difficult and time-consuming task because we must experiment with various approaches in a trial-and-error process. We must choose how many and what types of layers to use, which order to put layers in, and how many nodes to include in each layer, among other things to consider.

2.1.9 Region-based Convolutional Neural Network (R-CNN)

Girshick et al. [9] used the benefits of the convolutional neural network to create a network for object detection by combining region proposals with CNNs. They called it a region-based convolutional neural network, or R-CNN [9]. It takes a deep Neural Network trained originally for image classification using millions of annotated images and modifies it for object detection. The method generates around 2000 class-independent region proposals for the input image using a Selective Search algorithm. Then, warped all the regions to the same size since the CNN only accepts fixed-size input. Thus, these regions are scaled to the fixed size required by CNN. Next, using a CNN, each feature is computed for each of the proposed regions, and category-specific linear SVMs are used to classify each region [9].

Selective Search

It is an algorithm to generate region proposals to locate all the objects in an input image [18]. Uijlings et al. [18] consider the following variety of strategies to deal with to make an accurate selective search algorithm:

- Capture All Scales: Objects occur in different scales in an image, and the selective search algorithm should consider all object scales.
- Diversification Lighting conditions, colour, and more factors could influence how regions form an object, So a diverse set of strategies would work better as a single

strategy to deal with all cases.

- Fast to Compute The algorithm should be reasonably fast to create all the sets of possible object locations.

2.1.10 Fast Region-based Convolutional Neural Network (Fast R-CNN)

As a successor to R-CNN, Fast R-CNN builds on previous work that used deep convolutional networks to classify object proposals efficiently [8]. The training process in R-CNN is a multi-stage pipeline, which makes training expensive in terms of space and time, and it takes 47s with VGG16 to detect objects in a single test image [8]. So, in order to improve the training process of R-CNN, Girshick [8] developed Fast R-CNN, a faster object detection algorithm, in 2015.[8]

The network of Fast R-CNN works as follows:

- An input image is fed into a single CNN with multiple convolutional layers to generate a convolution feature map in Fast R-CNN.[8]
- The regions of interest are still generated using the Selective Search algorithm and fed into Fast R-CNN. Each proposal for a region is warped and fed into an RoI pooling layer.[8]
- The RoI pooling layer generates a fixed-length feature vector from the feature map for each object proposal and feeds it into a series of fully connected layers.[8]
- Finally, the fully connected layers are divided into two branches: the softmax classifier predicts the proposed region's class, and the regression output provides the four offset values for adjusting the bounding boxes.[8]

Fast R-CNN network is better than R-CNN and SPP networks [8], and the following are some advantages of Fast R-CNN over R-CNN:

- Fast R-CNN involves only training one CNN for an entire image rather than training multiple CNNs for all of the region proposals generated for an image [8].
- Multiple class-specific SVMs are replaced by a single softmax classifier, allowing the neural network extension without training a new module [8].
- When compared to R-CNN, the above factors make training single-stage faster.

The testing speed also significantly improves. According to Girshick [8], the speed of Fast R-CNN for training the VGG16 network has improved nine times than that of R-CNN. Similarly, the test speed for Fast R-CNN has increased 213 times when compared to R-CNN [8].

- Furthermore, Fast R-CNN improved detection accuracy by achieving an mAP score of 68.4 % for the PASCAL VOC 2012 dataset [8].

2.1.11 Faster Region-based Convolutional Neural Network (Faster R-CNN)

Faster R-CNN is an enhanced version of Fast R-CNN and uses a Region Proposal Network (RPN). Fast R-CNN reduces training and testing time, but the Selective Search algorithm used to generate region proposals is a time-consuming and computationally expensive method of producing region proposals [17]. As a result, Ren et al. [17] replaced Selective Search with Region Proposal Network (RPN) in an object detection algorithm known as Faster R-CNN in 2015. A region proposal network generates a fixed number of region proposals. Faster R-CNN combines RPN and Fast R-CNN to detect objects.

The network of Faster R-CNN works as follows:

- A stage for generating features for these objects from an input image fed to a CNN network [17].
- A n*n window is slid over the feature map obtained by the last convolution layer, mapping it to a lower-dimensional (e.g, 256-d) feature vector [17].
- The feature vector output is connected to two fully connected layers, one for classification and one for regression. RPN generates a set of k region proposals with varying aspect ratios for each sliding window location. Anchor boxes are the k regions proposed by the sliding window [17]. Each anchor box is associated with an objectness score and four bounding box coordinates. Anchor boxes are classified as object classes or backgrounds using the objectness score [17].

Because RPN generates a maximum of k regions for each location, the classification layer produces 2k scores to predict whether each region is an object or not, and the regression layer produces 4k corresponding to the four coordinates of each region. Following the detection of all anchor boxes, the relevant region proposals are chosen by applying a threshold to the objectness score. In essence, regions with an objectness score greater than a certain threshold are chosen and fed into the Fast R-CNN detector, along with a convolutional feature map [17].

The region proposal step is almost free because RPN shares the CNN feature map with the detection network [17]. Moreover, Faster R-CNN achieved state of the art accuracy with an mAP score of 70.4% on the PASCAL VOC 2012 dataset, and an mAP score of 73.2% on the PASCAL VOC 2007 dataset [17].

2.1.12 Convolutional Neural Networks training procedure

Huge annotated images could be required for successful CNN network training to learn different patterns, which could help the model to converge and predict well on unseen data. Having well-annotated training data can help tune the initial weights by updating their values in different layers while learning from the error and attempting to keep it as small as possible. After that, we use the test set to perform evaluations. The network adjusts the filter weights during the training process using a backpropagation algorithm. Backpropagation consists primarily of four stages: forward pass, loss function calculation, backward pass, and weight update. At the start of a training process, all weights are randomly initialized.
The forward pass involves running an input image from the training set through the network to generate an output. We use the predicted and generated outputs to calculate the loss function. We determine which weights to adjust and how much to reduce the loss function based on the loss function, and this stage is the "backward pass". Then, in the final stage of the weight update, we take and update all the weights according to the previous action. The following formula is used to update all of the weights:

$$w = wi - \lambda dL/dW \tag{2.1}$$

where w = New weight, wi = Initial weight,

λ = Learning rate, dL/dW = Derivative of loss function

The parameter λ is known as the learning rate. The user chooses the learning rate, which is an essential part of the training process. Because the weights are randomized at the start, the learning rate is usually kept high. A faster learning rate implies that the model could converge or minimize the loss function in less time. One iteration of learning consists of the four steps listed above. We carry out a fixed number of training steps to reduce the error function, which is the aim of the training process. The obtained model is then evaluated using the test dataset.

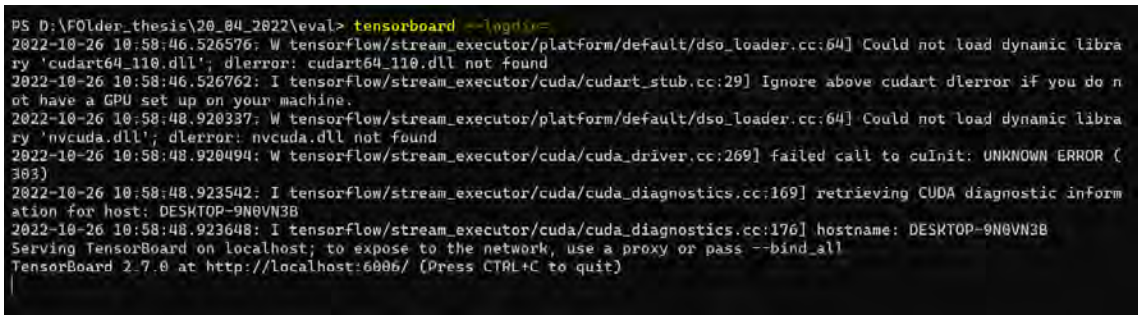
2.1.13 Tensorflow Object Detection API

The TensorFlow Object Detection API [4] is an open-source framework and a method for resolving object detection challenges [4], which offers easy-to-use pre-trained models for object detection tasks. This API makes the training, evaluating, and deploying of object detection models easier through available instructions and code examples for fine-tuning models to tackle different object detection problems. TensorFlow can be used for various purposes, but its main focus is deep neural network training and analysis. It supports easy deployment on CPUs, GPUs, TPUs, and computational device clusters and is designed for major platforms such as macOS, Windows, Linux, Android, and iOS. Tensors, which are multidimensional typed arrays, form the basis of TensorFlow computations. Mathematical operations are carried out at a tensor's node, whereas input-output operations are carried out on the tensor's edges. We find in TensorFlow Object detection API that different pre-trained models have been trained on different datasets. We chose for our work Faster R-CNN with ResNet 101. The model was trained using the COCO 2017 Dataset [13], which consists of 2.5 million labelled instances in 328 000 images, containing 91 object types [13].

Tensorboard

TensorBoard is a TensorFlow visualisation tool used to inspect and analyse graphed representations of how TensorFlow runs. Metrics such as mean Average Precision, training loss, and evaluation loss are included in the visualisations. TensorBoard allows to evaluation models performance through the time plane, making it easier to analyse a model as it evolves. We can run the Tensorboard during the model training to check whether the training is going correctly [4].

We use the command shown in Figure 2.1.11 to access Tensorboard from the terminal, and an example from our work with Tensorboard is shown in Figure 2.1.12.



```
PS D:\FOlder_thesis\20_04_2022\eval> tensorboard --logdir=
2022-10-26 10:58:46.526576: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2022-10-26 10:58:46.526762: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
ot have a GPU set up on your machine.
2022-10-26 10:58:48.920337: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'nvcuda.dll'; dlerror: nvcuda.dll not found
2022-10-26 10:58:48.920494: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (
303)
2022-10-26 10:58:48.923542: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic inform
ation for host: DESKTOP-9N0VN3B
2022-10-26 10:58:48.923648: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP-9N0VN3B
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.7.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Figure 2.1.11: Terminal command for Tensorboard

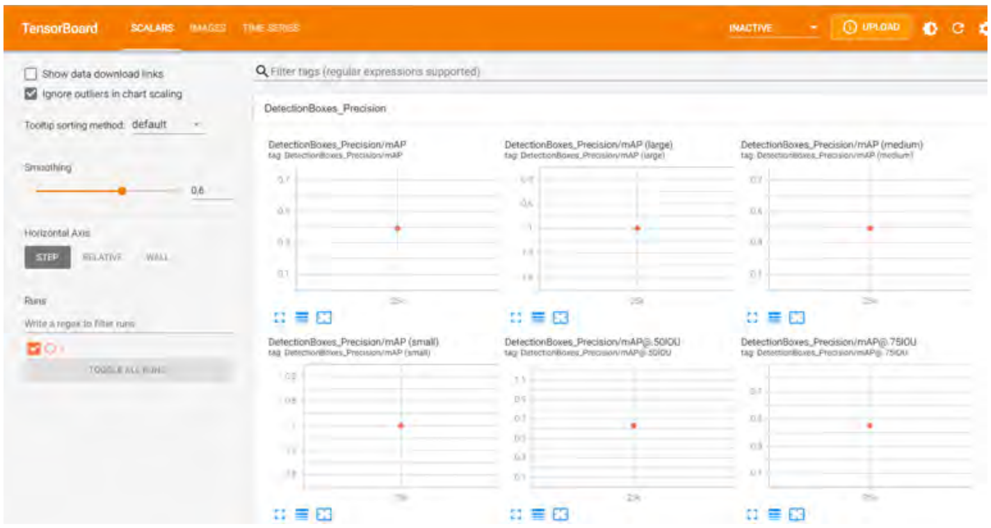

TensorBoard
SCALARS
IMAGES
TIME SERIES
INACTIVE
UPLOAD
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
Filter tags (regular expressions supported)
DetectionBoxes_Precision
DetectionBoxes_Precision/mAP
DetectionBoxes_Precision/mAP (large)
DetectionBoxes_Precision/mAP (medium)
DetectionBoxes_Precision/mAP (small)
DetectionBoxes_Precision/mAP@.50IOU
DetectionBoxes_Precision/mAP@.75IOU

Figure 2.1.12: Tensorboard

Chapter 3

Related Work

3.1 Introduction

Object detection is a critical problem in computer vision, with applications ranging from self-driving cars to security systems. Object detection attempts to locate and classify objects in an image or video. Object classification, tracking, and localization are three key research areas in object detection. In this literature review, we examined the state-of-the-art methods for object detection inside refrigerator papers and their strengths and limitations.

3.1.1 Smart Refrigerator inventory management using convolutional Neural Networks

Tae-Ho Lee et al. [12] developed an automatic inventory management system for intelligent refrigerators based on Deep Learning and Computer Vision. They divided the work into object detection and object tracking. Tae-Ho Lee et al. used YOLOv3 as an object recognition model to recognize 25 types of objects, including various fruits and drinks stored in the refrigerator. The system achieved an accuracy of 84.63% for a range of test cases. The dataset consisted of a small set of manually labelled images in combination with automatically generated images, using an automatic labelling method based on object segmentation and augmentation. They consider the effects of 3-dimensional viewing angle and illumination when creating a training dataset to achieve high accuracy in Deep Learning-based object detection. Wide-angle cameras mounted on the side of the refrigerator enabled the detection and tracking of incoming and outgoing objects in a refrigerator with an additional camera to detect variations in objects inside the refrigerator. By using multiple cameras, they found that object detection accuracy has improved, which also helps solve the occlusion problem when objects are occluded by human hands while moving.

3.1.2 Research on Food Recognition of Smart Refrigerator Based on SSD Target Detection Algorithm

Gao et al. [7] investigated how to effectively and accurately identify the food inside the refrigerator without opening the refrigerator. They compared popular algorithms such as R-CNN, Fast-R-CNN, Faster-R-CNN, YOLO, and SSD. They concluded that the SSD target detection algorithm is the most suitable for the current situation inside the refrigerator. They found that SSD512 is better than SSD300 in detecting food inside the refrigerator after performing a comparison test with the VOC2007 dataset. Based on the experimental analysis, SSD512 performed better than SSD300 on target detection, with an accuracy of 76.8% over 74.3%.

3.1.3 Object Detection in IoT Based Smart Refrigerators Using CNN

Ashwathan et al. [3] have developed an intelligent system that detects 14 items in the fridge every time an object is placed or removed from the fridge and calculates their quantity. The system also detects the presence of rotten fruits or vegetables inside the refrigerator to avoid food wastage. This intelligent system notifies the user if any object has run out of stock or has gone empty. The object detection module consists of the PIR sensor and a Pi camera connected to the Raspberry Pi microcontroller. Whenever a person opens the refrigerator, the PIR sensor detects motion and triggers the Pi camera to take an image of the refrigerator's contents. They used the YOLO model for the object detection part, which performed well with an accuracy of 95%.

3.1.4 Object Detection and Recognition for Visually Impaired Users: A Transfer Learning Approach

Wei-Cheng Won et al. [19] applied transfer learning to Single Shot Detector (SSD) and Faster R-CNN pre-trained models to detect and recognise 40 classes of ordinary objects in their surroundings and then compared their performances. Faster R-CNN performed better than the SSD model by achieving an mAP score of 0.8961 with an inference time of 7.2 seconds, while the SSD achieved an mAP score of 0.5708 with an inference time of 1.8 seconds. This work's purpose is to make the navigation of visually impaired people easier by identifying objects in their surroundings. Wei-Cheng Won et al. stated that resizing the input images to a fixed size of 300*300 pixels improved the training efficiency. Besides, applying dropout and data augmentation prevented the model from overfitting.

3.1.5 Object detection in refrigerators using Tensorflow

Kirti Agarwal's [1] dissertation focuses on detecting eight objects found within a refrigerator. He employs the Tensorflow Object Detection API to train and evaluate three models: SSD, Faster R-CNN, and R-FCN. He examines the models both as pre-trained and as fine-tuned. The fine-tuning process utilized a training dataset for eight food classes extracted from the ImageNet database. Faster R-CNN demonstrated the highest performance on the test food dataset derived from the ImageNet database with an mAP score of 81.74%, followed by R-FCN with an mAP of 80.33% and SSD with an mAP of 76.39%. However, SSD required significantly less detection time than the other two models; thus, Agarwal selected the SSD model for deploying object detection on mobile devices, as he considered the speed of an object detector as a crucial aspect of real-time applications.

3.1.6 End-to-End Airplane Detection Using Transfer Learning in Remote Sensing Images

Zhong Chen et al. [6] developed an aeroplane detection algorithm based on a single convolutional neural network through transfer learning. The aeroplane samples were collected from Google Earth and manually annotated. In the test phase, Zhong Chen et al. proposed a method for dealing with remote sensing images of large and different sizes of input images, adding MultiBlock and MapBlock layers. A MultiBlock layer is added before the input layer, and the MapBlock layer is added after the detection output layer, which aims to effectively solve the problem of slight object loss caused by directly resizing the image. The SSD method achieved an mAP of 86.28%. On the other side, adding MultiBlock and MapBlock layers has improved the model accuracy to 96.23% on the test set.

3.1.7 Object Detection Using Deep CNNs Trained on Synthetic Images

P.S. Rajpura et al. [16] applied transfer learning to a Fully Convolutional Network (FCN) detector trained almost entirely on synthetic images for detecting packaged food products clustered in refrigerator scenes. They analysed the influence of factors like training data set size and 3D model dictionary size on the detection performance. They found that CNN fully trained with 4000 synthetic images achieved a 24 mAP and underperformed against one with 400 real images, which achieved a 28 mAP. However, adding 4000 synthetic images to the real dataset boosts the detection performance by 12% (achieved 36 mAP).

We have created Table 3.1.1 to provide a summary of the papers we reviewed. The table is presented below:

3.2 Summary

This chapter summarises seven papers focusing on detecting objects inside refrigerators and other related domains. Each paper uses different concepts and strategies and offers new ideas. After reviewing the papers, the most used models for detecting objects in fridges are the Faster R-CNN, SSD, and R-FCN. Most use the transfer learning concept to fine-tune an existing, pre-trained model. In our work, we will use the Faster R-CNN model to perform the detection task in our fridge. In the next chapter, we will go through the necessary steps to develop and evaluate the object detection system.

Year	Title	Description	Result
2022	Object Detection in IoTBased Smart Refrigerators Using CNN [3]	Target: Create a system in the refrigerator to detect different products based on quantity and food quality monitoring module using sensors such as Pi Camera, PIR motion sensor, ultrasonic sensor, and gas sensor. Models: YOLO Datasets: Set of 500 images downloaded from Kaggle	Accuracy of YOLO: 95%
2021	Object Detection and Recognition for Visually Impaired Users: A Transfer Learning Approach [19]	Target: Use transfer learning on two pre-trained models to detect and recognise 40 classes of ordinary objects in surroundings and compare the performance of the two models. Models: Single Shot Detector Mobilenet V1 (SSD) and Faster Region-Convolutional Neural Network Inception V2 (Faster R-CNN) Datasets: The training included 29 object classes from the COCO dataset + 11 additional classes.	The mAP of Faster R-CNN: 0.8961 with an inference time of 7.2 seconds The mAP of SSD: 0.5708 with an inference time of 1.8 seconds
2019	Research on Food Recognition of Smart Refrigerator Based on SSD Target Detection Algorithm [7]	Target: To detect food inside the refrigerator through the SSD target detection algorithm, which is based on the regression of deep convolutional neural network Models: SSD512 and SSD300 Datasets: Source of experimental pictures and VOC2007 dataset	The mAP of SSD300: 74.3% The mAP of SSD512: 76.8%
2018	Object detection in refrigerators using Tensorflow [1]	Target: It focuses mainly on detecting eight objects in a refrigerator by comparing three models and selecting the most suitable one for this problem. Models: SSD-MobileNet-v2, Faster R-CNN-ResNet-101 and R-FCN-ResNet-101 Datasets: ImageNet	The mAP of SSD-MobileNet-v2: 76.39% The mAP of Faster R-CNN-ResNet-101: 81.74% The mAP of R-FCN-ResNet-101: 80.33%
2018	End-to-End Airplane Detection Using Transfer Learning in Remote Sensing Images [6]	Target: Use deep learning and transfer learning-based object detection technology to detect aeroplanes in remote sensing images. Models: SSD-VGG16 for the training phase and add Multiblock and Mapblock layers for testing phase Datasets: Is collected from satellite images of the world's top 30 airports in Google Earth	The mAP of SSD-VGG16: 86.28% with an average prediction time 513.76 ms for each image The mAP after adding Multiblock and Mapblock layers: 96.23% with an average time 1934.63 ms for each image
2017	Object Detection Using Deep CNNs Trained on Synthetic Images [16]	Target:Using transfer learning to train an object detector almost entirely on synthetically rendered datasets to detect packaged food in refrigerator scenes Models: Fully Convolutional Network Detector (FCN) Datasets: ShapeNet Dataset	The mAP of the detector: 24 on a test set with 55 distinct products as objects of interest and 17 distractor objects.
2014	Smart Refrigerator Inventory Management Using Convolutional Neural Networks [12]	Target: Build a deep learning-based system for object recognition and inventory management Models: YOLOv3 Datasets: Small set of manually labelled images + automatically generated images	The mAP of the YOLOv3: 84.63% for a set of test cases

Table 3.1.1: Summary of the Literature Review

Chapter 4

Implementation

This thesis aims to train and fine-tune one of the state-of-the-art object detectors, specifically the Faster R-CNN, on a custom dataset. The objective is to test different calibrations on the data to determine which approaches should be applied to detect four classes of interest in different ways and at different positions. Additionally, the study aims to investigate the ability of the Faster R-CNN to detect occluded objects and then generalize to detect more classes in future work. This chapter describes the dataset used in the research and the required pre-processing steps. The object detection model (Faster R-CNN) pre-trained in Tensorflow is then fine-tuned. Finally, we describe the evaluation metrics used in this study to objectively measure our model's performance.

4.1 Data Collection and Preprocessing

Training deep convolutional neural networks (CNN) is challenging in computer vision because it requires collecting many labelled training images to train image classifiers and object detectors. Moreover, training an object detection model from scratch requires a lot of GPUs and a huge number of labelled images, and it is also time-consuming. In this thesis, we fine-tuned a pre-trained object detection model from the TensorFlow 2 Detection Model Zoo [1], trained initially on the COCO 2017 dataset. Our goal is to detect four objects, namely, a bell pepper, butter, tomato, and carrot, on the top shelf of our refrigerator. The raw data consisted of frames extracted from the video footage of our refrigerator. Many pictures were included in different ways and positions

[1]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

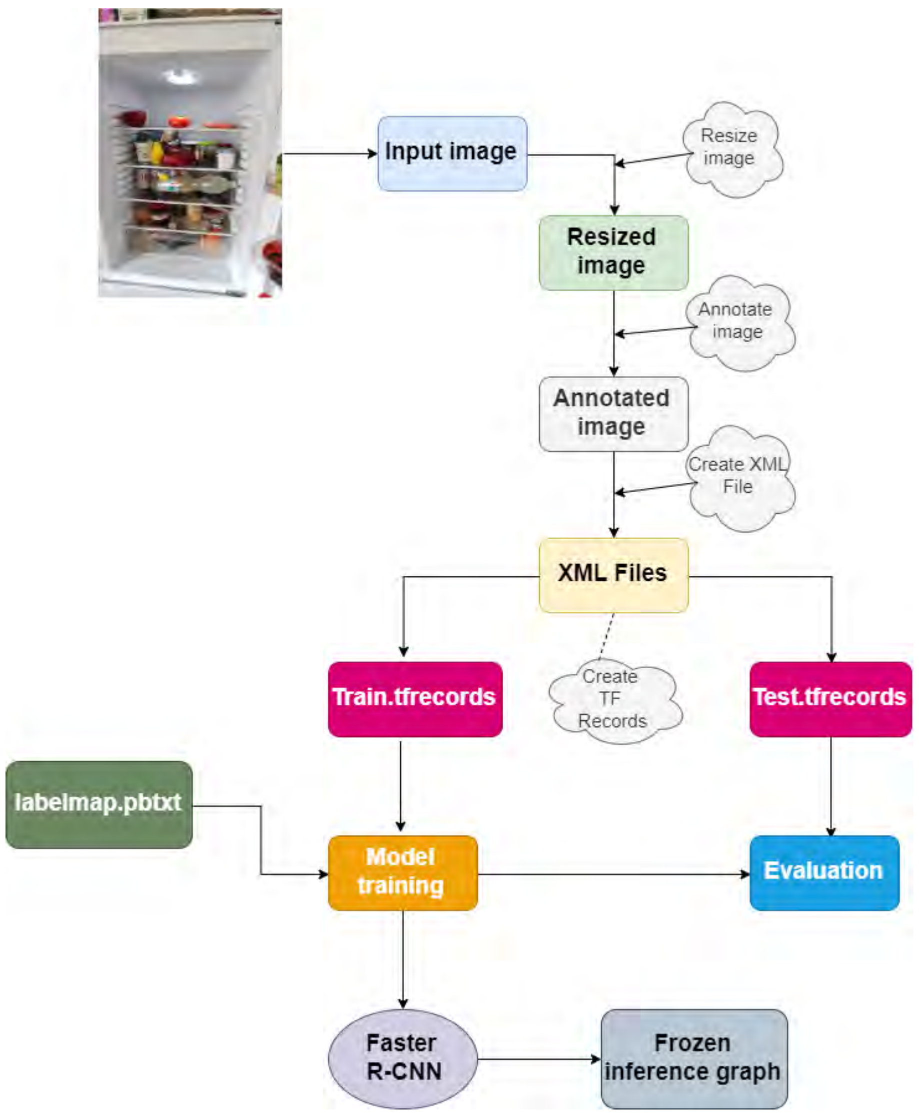


Figure 4.0.1: Implementation of the Object Detector Training and Evaluation Process

to investigate the effects of specific factors, such as the color and shape of the object classes. The training data was taken with our camera phone and consisted of refrigerator images for each object class. The test data consists of refrigerator frames with all four objects combined in different positions and ways on the upper shelf to ensure the test datas independence from the training data. The final data consisted of 400 training data images and 100 test data images, and we pre-processed the gathered datasets for training and testing purposes as follows:

- Resizing of images: We created an automated script to resize the images of the training and test datasets to a width of 400 and a height of 640 using the OpenCV library cv2.resize, which gave good results in detecting the objects in the test dataset.
- Annotation of Datasets: We annotated around 1200 images and kept the best 500 for training and testing purposes. The annotation part of the manual labeling of images required significant time and effort. We employed "LabelImg" for annotating the graphical images. This tool, based on Python, facilitates the labelling process through bounding box coordinates, which are then saved in either PASCAL VOC or YOLO format. Our work utilized the PASCAL VOC format to maintain the annotations via LabelImg.
- Create training and test dataset: After completing the annotation process, we divided the dataset into two sets: the training dataset and the test dataset.
- Create TFRecords: To fine-tune a pre-trained TensorFlow object detection model, we must convert the training and test datasets into a TF Record format using a ready-made Python script, generate_tfrecord.py. TFRecord is a commonly utilized format by TensorFlow for training object detection models. The original training and testing datasets were in XML file format, which were then transformed into the TFRecord format using the following Python command:

```
python generate_tfrecord.py
  —x path/to/train(test)_labels.xml
  —l path/to/label_map.pbtxt
  —o path/to/train(test).record
```

  Following the execution of the code shown above, two new files should be created and saved under the output paths specified in the script: "path/to/train.record" and "path/to/test.record", respectively.

4.2 Configuration of a Training Job

After reviewing much research literature, we decided to go with the Faster R-CNN with ResNet-101 model since it is the most accurate model based on the comparison results obtained in the literature review made by Agarwal [1] between SSD with MobileNet-v2, Faster R-CNN with ResNet-101, and R-FCN with ResNet-101. Thus, we decided to work on this model and extract some factors that could affect the model's accuracy and performance, which could be helpful for future work to consider those factors while performing object detection tasks.
After implementing all the steps above, the next step would be starting the fine-tuning process of the faster_rcnn_resnet101_v1_640*640_coco17 model, which provides a relatively good performance in detecting objects in the COCO 2017 dataset.

4.2.1 Download Pre-trained Model

The pre-trained Faster R-CNN with ResNet-101 model is available in the TensorFlow 2 Detection Model Zoo, where it can be downloaded. The download in the format *.tar.gz should begin by clicking on the model's name. After completing the download step, we opened it with a decompression program (we used WinZip). After that, we opened the *.tar folder that should be visible when we opened the compressed folder and extracted its contents into the training demo/pre-trained models folder. Because we downloaded the Faster R-CNN model, the training_demo directory should now look as follows:

```
training_demo/
-- pre-trained-models/
  -- faster_rcnn_resnet101_v1_640x640_coco17_tpu-8/
    - checkpoint
    - saved_model
    - pipeline.config
```

4.2.2 Training pipeline configuration

We now need to create a new directory for the training job, which we named my_faster_rcnn_resnet101 under the training_demo/models path. We copy the pipeline.config file from the pre-trained-models directory and paste it inside the newly created model directory. The model's directory should look like this:

```
training_demo/
 - ...
```

```
  - models/
    - my_faster_rcnn_resnet101/
      - pipeline.config
```

After this step, we implemented the changes shown in Figure 4.2.1 on the pipeline.config file, which is the basis of our fine-tuning. Afterwards, we could start the training job on our custom-gathered fridge dataset.

4.2.3 Training the model

The model requires the model_main_tf2.py [2] file in the training_demo/ directory. Next, we need to run the following script to be able to initiate the training of our model:

```
python model_main_tf2.py

—model_dir=/.../training_demo/models/my_faster_rcnn_resnet101

—pipeline_config_path=/.../my_faster_rcnn_resnet101/pipeline.config
```

The script includes the pipeline.config file and the model directory created for my_faster_rcnn_resnet101.

Once the training is successfully started, it will look similar to Figure 4.2.2.

Utilizing the computational resources of a Denethor, an Ubuntu-based computer equipped with a powerful NVIDIA Corporation GV100 [TITANV] graphics card located within the FORWISS sub-network provided by the University of Passau, significantly reduced the training time of a Faster R-CNN model, with a typical training duration of 45 minutes and 28 seconds. However, various factors such as the hardware's computational power, the utilization of a Tensorflow GPU or CPU, the complexity of the objects to be detected, and other considerations, may affect the training time.

4.3 Evaluation of the model

Periodically, checkpoint files were generated within the training directory, which served as snapshots of the model's state at various stages of the training process. These checkpoint

[2] https://github.com/tensorflow/models/blob/master/research/object_detection/model_main_tf2.py

files were subsequently employed in the evaluation phase to determine the model's proficiency in detecting objects within the test dataset. The evaluation outcomes were subsequently encapsulated in metrics that allow for the monitoring of progress over time. To achieve our objectives, we had to undertake the following evaluation procedure:

- We must install and download the desired metric, as well as install the pycocotools package, which is a dependency of the Object Detection API, and place it in the /.../models/research directory using the following command:
- We need to acquire the desired metric and install the pycocotools package, which is a dependency of the Object Detection API. This should be done by placing the package in the /.../models/research directory and using the command:

```
git clone https://github.com/cocodataset/cocoapi.git
cd cocoapi/PythonAPI
make
cp —r pycocotools <PATH_TO_TF>/TensorFlow/models/research/
```

- We need to set metrics_set to "coco_detection_metrics" in the pipeline.config file, as shown in Figure 4.3.1
- We run after that the following script to start the evaluation process:

```
python model_main_tf2.py
——model_dir=models/my_faster_rcnn
——pipeline_config_path=models/my_faster_rcnn/pipeline.config
——checkpoint_dir=models/my_faster_rcnn
```

During the evaluation process, the system will continually monitor the most recent models/my_faster_rcnn_resnet101/ckpt-* checkpoint files with a default interval of 300 seconds, to assess the performance of the model. The results will be saved as TF event files (events.out.tfevents.*) in the models/my_faster_rcnn_resnet101/eval_0 directory. These files can be utilized to analyze the computed metrics, as outlined in the subsequent section.

4.3.1 Visualize results using Tensorboard

TensorFlow provides an efficient mechanism for monitoring and visualizing various training and evaluation metrics during our model's training. As previously discussed, Tensorboard is the tool that enables this functionality. To evaluate the performance of our model during the evaluation phase, we can utilize the following steps: Navigate to the Eval file directory and execute the command shown in Figure 4.3.2.

This will initiate Tensorboard in the current directory. The results of this action are depicted in Figure 4.3.3 and will display the contents of the eval file directory.

Then, open a browser and type http://localhost:6006/ in the address bar. A dashboard will be displayed, similar to the one shown in Figure 4.3.4 below.

4.4 Evaluation metrics

Object detection involves identifying target objects within an image or video by localizing them and drawing bounding boxes around them. Several metrics can be used to evaluate the performance of an object detector. One of the most common is average precision (AP). In our work, we used mean average precision (mAP) and average recall (AR) to assess the performance of our model.

The confusion matrix table, shown in Figure 4.4.1, is commonly used to assess the performance of a classification algorithm. However, in the object detection domain, we could use the confusion matrix to compute various evaluation metrics, such as precision and recall. We will go over the elements of the confusion matrix and their definitions:

- True positives (TP): The object detector could correctly detect an object in the image and surround it with a bounding box.
- True negatives (TN): These are the instances where the object detector correctly identified the absence of an object in the image and did not place a bounding box around it. Regardless, this does not apply because we have infinite bounding boxes that should not be detected within any given image. [14]
- False positives (FP): These are instances where the object detector incorrectly identified the presence of an object in the image and surrounded it with a bounding box.
- False negatives (FN): These are cases where the object detector incorrectly identified the absence of an object in an image and failed to draw a bounding box around it.

4.4.1 Precision

Precision is the percentage of correct positive predictions made out of all predictions. In other words, the model's ability to identify only relevant objects. [14]
The following equation expresses the definition of precision:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \tag{4.1}$$

4.4.2 Recall

It is the fraction of objects of interest detected by the system out of the total number of objects of interest in the image or video. For example, if an object detection system detects 50 out of 100 objects of interest, the recall is $50/100 = 0.5$. [14]
The equation for the recall is as follows:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \tag{4.2}$$

Precision and recall are important metrics for evaluating the performance of object detection models. When one of these metrics improves, the other tends to decrease. For instance, a model with a higher detection threshold will have fewer false positive detections but will also retrieve fewer accurate positive detections. So, as the threshold increases, precision will improve while recall decreases. In any application, it is necessary to find a balance between precision and recall that meets the application's specific needs. In our case, having fewer false positives and, thus, higher precision is essential.

4.4.3 Intersection Over Union (IoU)

IoU (Intersection over Union) is a metric used to evaluate the performance of object detectors. It represents the degree of overlap between the predicted bounding box and the actual ground truth region, assessing the prediction's accuracy. The IoU score is calculated using the following formula:

$$IoU = \frac{Area of Intersection (Overlap)}{Area of Union} \tag{4.3}$$

Figure 4.4.2 displays the calculation of the IoU score between two bounding boxes, the predicted and the ground truth. This score is obtained by estimating the areas of both the overlap and union of the bounding boxes. A high IoU score, approaching 1, demonstrates a significant overlap between the two bounding boxes, while a low IoU score, near 0, suggests limited overlap.

4.4.4 Average Recall (AR)

The average recall measures a system's ability to detect objects of interest in an image or video. It is defined as the average of all recall rates in a dataset across all objects of interest. The average recall is often used with other evaluation metrics, such as precision and the F1 score, to understand a model's performance better. It is beneficial when working with imbalanced datasets, where some classes have significantly more instances than others. By calculating the average recall across all classes, we can see how well the model can detect instances of all classes rather than just the most commonly occurring ones.

4.4.5 Recall-Precision curve

The recall-precision curve is a graphical representation of a model's precision at various levels of recall. It is created by plotting the precision of the model on the y-axis and the recall on the x-axis. The curve is generated by varying the decision threshold for making positive predictions and calculating each threshold's precision and recall values. A recall-precision curve is a helpful tool for evaluating the trade-off between precision and recall and for selecting the best model for a given application.

4.4.6 Average Precision

Average precision (AP) is a metric used to evaluate the accuracy of object detection models. It is calculated by first generating a precision-recall curve, which plots the precision (the proportion of true positive detections among all positive detections) against the recall (the proportion of true positive detections among all actual positive instances in the dataset). The average precision is then computed by taking the area under this curve and dividing it by the total number of recall levels. The mathematical expression for average precision (AP) is as follows:

$$AP = \sum_{r} (precision(r) - precision(r-1)) * recall(r)$$

where $precision(r)$ is the precision at recall level r, and $precision(r-1)$ is the precision at the previous recall level $(r-1)$. This equation sums the precision values at different recall levels and multiplies them by the corresponding recall values to calculate the area under the curve. The AP is then calculated by dividing this value by the total number of recall levels.

In object detection tasks, the AP is often used to evaluate the performance of a model on a particular class or set of classes, and the mean average precision (mAP) is calculated as the mean of the AP values for all classes being evaluated.

On the other hand, average recall is a metric that measures the proportion of true positive detections among all actual positive instances. It is calculated by taking the average recall values for all food categories separately. A high average recall value indicates that the object detection system is able to accurately identify a large proportion of the actual positive instances across all food categories.

4.4.7 Mean Average Precision (mAP)

The mean average precision (mAP) is a metric used to evaluate the accuracy of object detection models. It is calculated by taking the average precision of all object classes being evaluated, where precision is the area under the precision-recall curve for each class. The mean average precision is then calculated by taking the mean of the average precision values for all classes.

$$\textbf{mAP} = \frac{1}{N}\sum_{j=1}^{M} \mathrm{AP}_j, \tag{8}$$

AP_j represents the AP in the jth class, and M represents the total number of classes being evaluated. Average precision is the area under the recall-precision curve for each object class. In object detection tasks, the mAP is often used as a benchmark to compare the performance of different models and to select the model that performs the best. Mean average precision (mAP) is a metric that summarizes the overall performance of the system by averaging the average precision values for each object category. It takes into account both precision and recall and provides a single number that summarizes the performance of the system across all object categories. A high mAP value indicates that the system is able to accurately identify a large proportion of objects across all categories while maintaining a high level of precision.

4.4.8 Relation between mAP and AR

In general, mAP and average recall are related but measure different aspects of the system's performance. mAP considers the trade-off between precision and recall, while average recall focuses on the proportion of true positive detections among all actual positive instances.

When analyzing the results according to mAP and average recall (AR), it is essential to look at both metrics and consider their relationship. A high mAP value and a high average recall value generally indicate that the system is performing well overall and can accurately identify a large proportion of objects across all categories while maintaining a considers both precision and recall and providing a single number that summarizes the system's performance of its performance. For example, suppose mAP is high, but average recall can be that case, the system can identify a large proportion of objects across all categories with high precision but struggles to identify all actual positive instances.

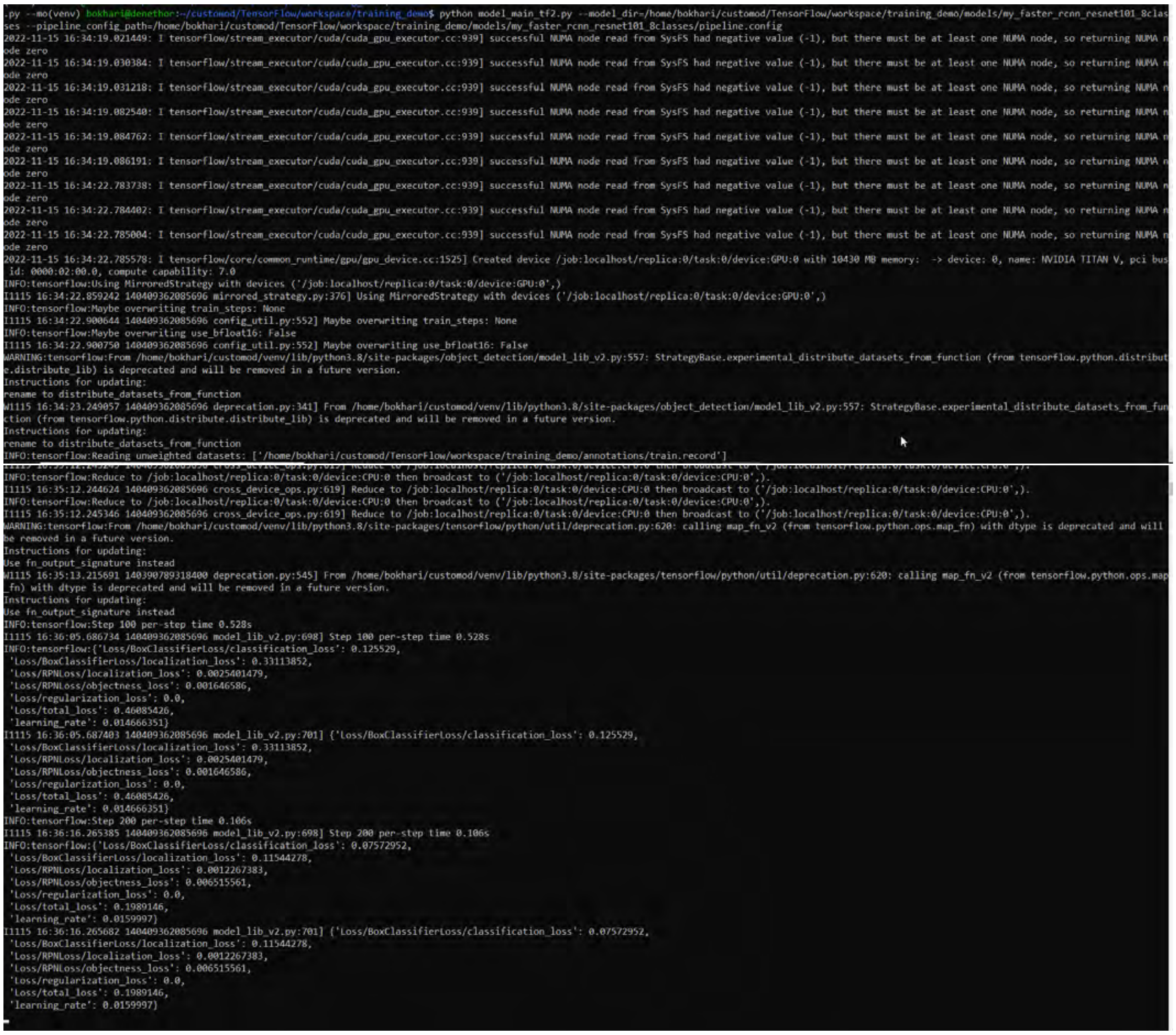
In summary, mean average precision (mAP) and average recall (AR) are both metrics used to evaluate the performance of object detection systems. They measure different aspects of the system's performance. A high mAP value and a high average recall value generally indicate that the system is performing well overall and can accurately identify a large proportion of objects across all categories while maintaining a high level of precision. Analyzing the results separately for each food category could help identify which categories the system is performing well in and which categories it is struggling with.

4.5 Summary

This chapter discussed using Tensorflow to implement object detection inside our refrigerator. We described collecting and preprocessing data specifically tailored to our needs. We explained the method for fine-tuning a pre-trained model, specifically the Faster-RCNN-ResNet101 model, using the Tensorflow object detection API. We also explored various configuration settings for fine-tuning this model. Finally, we covered the evaluation metrics used in the current study to assess the performance of our object detection models. The following chapter will further evaluate our object detection model using these metrics.

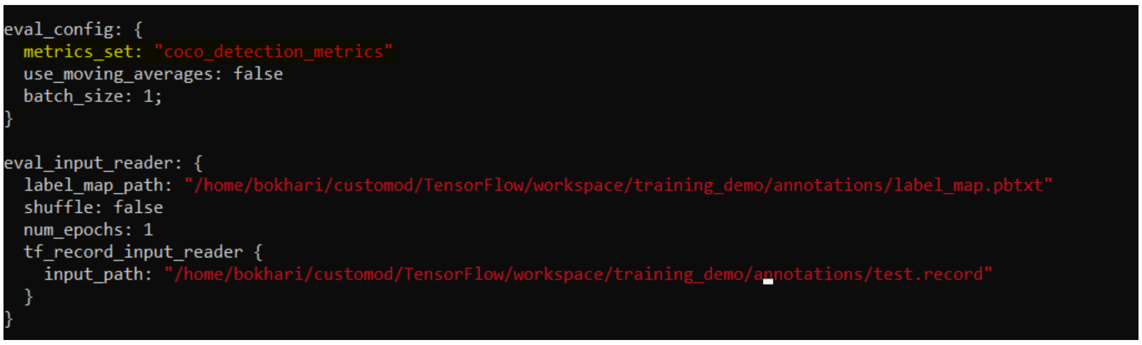
```
model {
  faster_rcnn {
    num_classes: 4                # Number of classes set to 4
    ...
    }
                                ...

train_config: {
  batch_size: 1                # batch_size set to 1
  sync_replicas: true
  startup_delay_steps: 0
                                ...

  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint: "/home/bokhari/customod/TensorFlow/workspace/training_demo/pre-trained
    -models/faster_rcnn_resnet101_v1_640x640_coco17_tpu-8/checkpoint/ckpt-0" # Updated path
  fine_tune_checkpoint_type: "detection"          # set to detection
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
                               ...
train_input_reader: {
  label_map_path: "/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/
    label_map.pbtxt"     # Define label_map_path to our path
  tf_record_input_reader {
    input_path: "/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/train.
    record"     # Give train.record path
  }
}
eval_config: {
  metrics_set: "coco_detection_metrics" # Used metrics_set
    ...
}

eval_input_reader: {
  label_map_path: "/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/
    label_map.pbtxt"               # Give the label_map_path for the evaluation step
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/test.
    record"            # Provide the test.record path for the evaluation step
  }
}
```

Figure 4.2.1: Modifications to the Configuration: An Overview of the Config File Changes



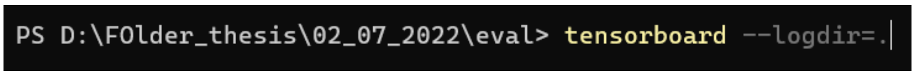
```
.py --mo(venv) bokhari@denethor:~/customod/TensorFlow/workspace/training_demo$ python model_main_tf2.py --model_dir=/home/bokhari/customod/TensorFlow/workspace/training_demo/models/my_faster_rcnn_resnet101_8clas
ses --pipeline_config_path=/home/bokhari/customod/TensorFlow/workspace/training_demo/models/my_faster_rcnn_resnet101_8classes/pipeline.config
2022-11-15 16:34:19.021449: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:19.030384: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:19.031218: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:19.082540: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:19.084762: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:19.086191: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:22.783738: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:22.784402: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:22.785004: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA n
ode zero
2022-11-15 16:34:22.785578: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 10430 MB memory:  -> device: 0, name: NVIDIA TITAN V, pci bus
 id: 0000:02:00.0, compute capability: 7.0
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0',)
I1115 16:34:22.859242 140409362085696 mirrored_strategy.py:376] Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0',)
INFO:tensorflow:Maybe overwriting train_steps: None
I1115 16:34:22.900644 140409362085696 config_util.py:552] Maybe overwriting train_steps: None
INFO:tensorflow:Maybe overwriting use_bfloat16: False
I1115 16:34:22.900750 140409362085696 config_util.py:552] Maybe overwriting use_bfloat16: False
WARNING:tensorflow:From /home/bokhari/customod/venv/lib/python3.8/site-packages/object_detection/model_lib_v2.py:557: StrategyBase.experimental_distribute_datasets_from_function (from tensorflow.python.distribut
e.distribute_lib) is deprecated and will be removed in a future version.
Instructions for updating:
rename to distribute_datasets_from_function
W1115 16:34:23.249057 140409362085696 deprecation.py:341] From /home/bokhari/customod/venv/lib/python3.8/site-packages/object_detection/model_lib_v2.py:557: StrategyBase.experimental_distribute_datasets_from_fun
ction (from tensorflow.python.distribute.distribute_lib) is deprecated and will be removed in a future version.
Instructions for updating:
rename to distribute_datasets_from_function
INFO:tensorflow:Reading unweighted datasets: ['/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/train.record']
[illegible]
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).
I1115 16:35:12.244624 140409362085696 cross_device_ops.py:619] Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).
I1115 16:35:12.245346 140409362085696 cross_device_ops.py:619] Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).
WARNING:tensorflow:From /home/bokhari/customod/venv/lib/python3.8/site-packages/tensorflow/python/util/deprecation.py:620: calling map_fn_v2 (from tensorflow.python.ops.map_fn) with dtype is deprecated and will
be removed in a future version.
Instructions for updating:
Use fn_output_signature instead
W1115 16:35:13.215691 140390789318400 deprecation.py:545] From /home/bokhari/customod/venv/lib/python3.8/site-packages/tensorflow/python/util/deprecation.py:620: calling map_fn_v2 (from tensorflow.python.ops.map
_fn) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Use fn_output_signature instead
INFO:tensorflow:Step 100 per-step time 0.528s
I1115 16:36:05.686734 140409362085696 model_lib_v2.py:698] Step 100 per-step time 0.528s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.125529,
 'Loss/BoxClassifierLoss/localization_loss': 0.33113852,
 'Loss/RPNLoss/localization_loss': 0.0025401479,
 'Loss/RPNLoss/objectness_loss': 0.001646586,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.46085426,
 'learning_rate': 0.014666351}
I1115 16:36:05.687403 140409362085696 model_lib_v2.py:701] {'Loss/BoxClassifierLoss/classification_loss': 0.125529,
 'Loss/BoxClassifierLoss/localization_loss': 0.33113852,
 'Loss/RPNLoss/localization_loss': 0.0025401479,
 'Loss/RPNLoss/objectness_loss': 0.001646586,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.46085426,
 'learning_rate': 0.014666351}
INFO:tensorflow:Step 200 per-step time 0.106s
I1115 16:36:16.265385 140409362085696 model_lib_v2.py:698] Step 200 per-step time 0.106s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.07572952,
 'Loss/BoxClassifierLoss/localization_loss': 0.11544278,
 'Loss/RPNLoss/localization_loss': 0.0012267383,
 'Loss/RPNLoss/objectness_loss': 0.006515561,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.1989146,
 'learning_rate': 0.0159997}
I1115 16:36:16.265682 140409362085696 model_lib_v2.py:701] {'Loss/BoxClassifierLoss/classification_loss': 0.07572952,
 'Loss/BoxClassifierLoss/localization_loss': 0.11544278,
 'Loss/RPNLoss/localization_loss': 0.0012267383,
 'Loss/RPNLoss/objectness_loss': 0.006515561,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.1989146,
 'learning_rate': 0.0159997}
```

Figure 4.2.2: Two start training examples



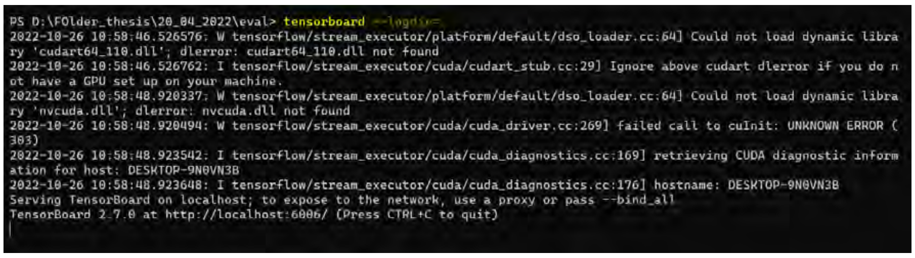
```
eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1;
}

eval_input_reader: {
  label_map_path: "/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/home/bokhari/customod/TensorFlow/workspace/training_demo/annotations/test.record"
  }
}
```

Figure 4.3.1: Metric used in our work



```
PS D:\FOlder_thesis\02_07_2022\eval> tensorboard --logdir=.
```

Figure 4.3.2: Command line for starting Tensorboard



```
PS D:\FOlder_thesis\20_04_2022\eval> tensorboard --logdir=
2022-10-26 10:58:46.526576: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2022-10-26 10:58:46.526762: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
ot have a GPU set up on your machine.
2022-10-26 10:58:48.920337: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
ry 'nvcuda.dll'; dlerror: nvcuda.dll not found
2022-10-26 10:58:48.920494: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (
303)
2022-10-26 10:58:48.923542: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic inform
ation for host: DESKTOP-9N0VN3B
2022-10-26 10:58:48.923648: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP-9N0VN3B
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.7.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Figure 4.3.3: Results of running the command

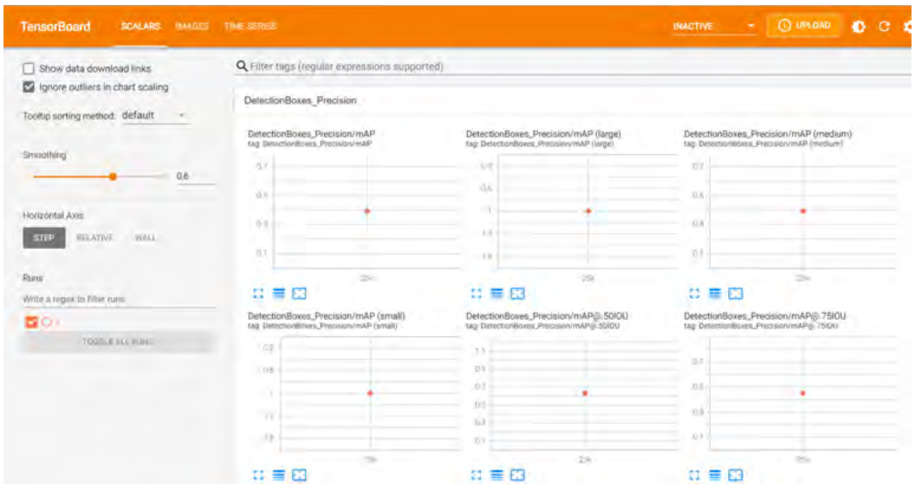
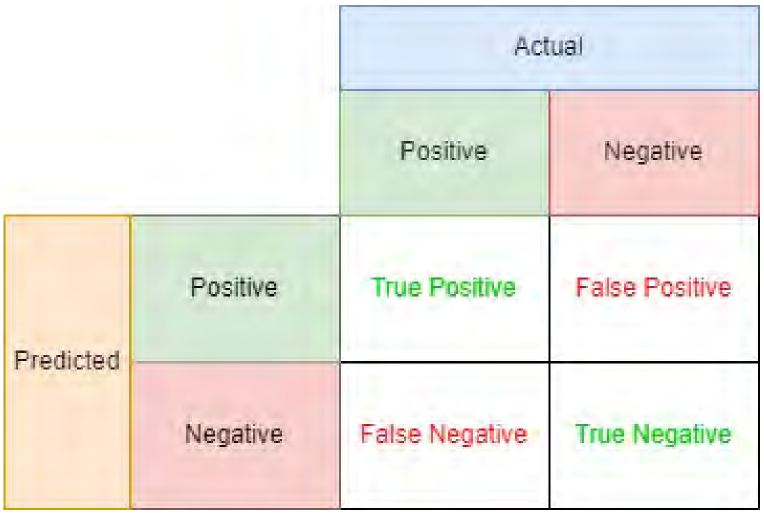

TensorBoard
SCALARS
IMAGES
TIME SERIES
INACTIVE
UPLOAD
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
Filter tags (regular expressions supported)
DetectionBoxes_Precision
DetectionBoxes_Precision/mAP
DetectionBoxes_Precision/mAP (large)
DetectionBoxes_Precision/mAP (medium)
DetectionBoxes_Precision/mAP (small)
DetectionBoxes_Precision/mAP@.50IOU
DetectionBoxes_Precision/mAP@.75IOU

Figure 4.3.4: Tensorboard dashboard example



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 4.4.1: Confusion Matrix

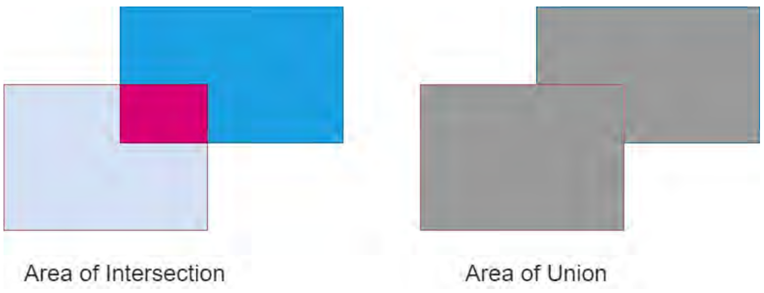


Figure 4.4.2: Areas of intersection and union

Chapter 5

Results

This chapter presents the results and discusses the performance of the fine-tuned object detector Faster R-CNN using the metrics mean average precision (mAP) and average recall (AR). We used two datasets gathered from different refrigerators under different conditions to train the Faster R-CNN object detection model. We used the TensorFlow Object Detection API [1] to train the model. Furthermore, we did the evaluation using several test datasets that were independent of the training dataset. We compared the detector's accuracy between the two datasets. Moreover, we assessed how well the detector could detect the four objects of interest on the top shelf of the refrigerator. We focused our research on various factors that address the dataset's quality and could affect the performance of an object detector based on related work gathered during the literature study. We applied those factors during the conduction of the research. Before starting the training phase, we had to prepare the datasets. The TensorFlow Object Detection API expects inputs to be in the TFRecord format. During preparation, we converted the datasets to the correct format, and specific modifications were made and explained in the previous section.

5.1 Data Preparation

We collected the dataset for this research by taking photos inside two different fridges under different conditions with our mobile phone camera. We created a Python script that helped us save frames from the video recordings inside our fridges to generalize more images and capture the objects from different angles. Furthermore, we also did the

[1] https://github.com/tensorflow/models/tree/master/research/object_detection

annotation manually, which was time-consuming and laborious.

5.1.1 Description of the first dataset

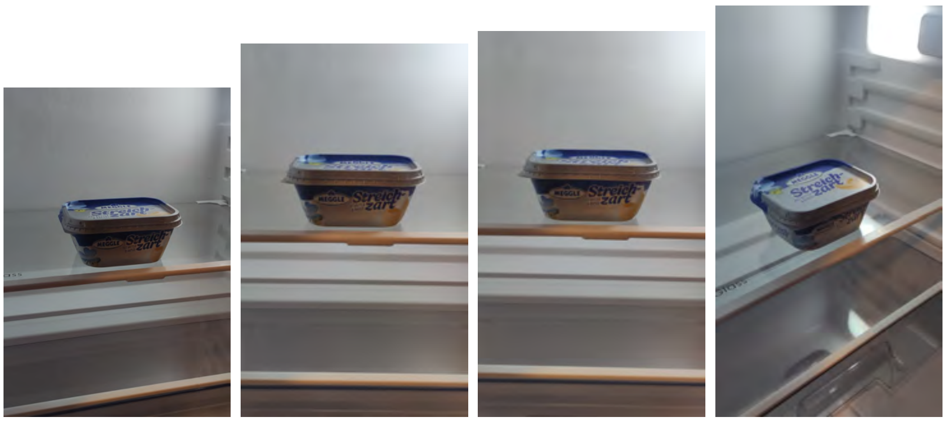


Figure 5.1.1: Training butter pictures examples from the first refrigerator

The first fridge was a small one inside a closed kitchen with poor luminosity conditions, thus leading to poor-quality images (see Figure 5.1.1). Some of the first dataset's characteristics include insufficient image variety per class and unbalanced training data. We took pictures for each object class, and in the test set, we combined the objects. Finally, we converted the images and their corresponding labels to TFRecord format.

5.1.2 Description of the second dataset

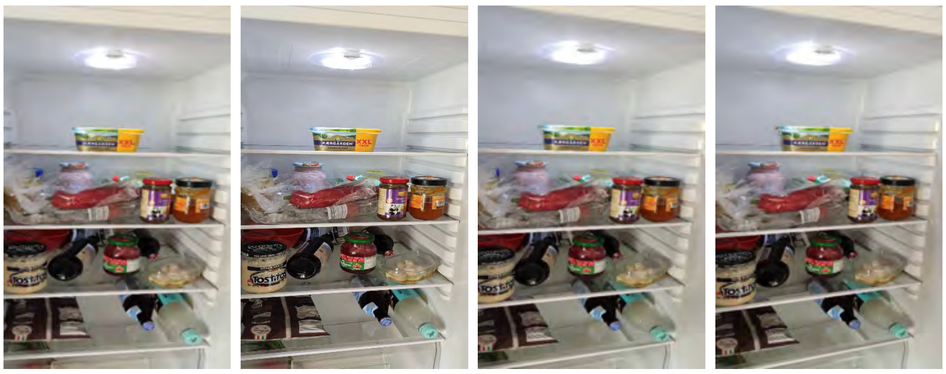


Figure 5.1.2: Training butter pictures examples from the second refrigerator

The second refrigerator was more oversized with better lighting conditions (see Figure 5.1.2), thus leading to better-quality images. In the training collection, we took various photos from different angles, far from and close to the fridge, for each class. In the test set, we combined the objects in different manners and orders to guarantee the independence of the training dataset from the test dataset. We also converted the images and their corresponding labels to TFRecord format.

We summarized the comparison between the two datasets in Table 5.1.1. We addressed the significant differences between the two datasets and evaluated both to see if the differences could have a significant impact on object detector accuracy.

Table 5.1.1: Comparison between datasets

First Dataset	Second Dataset
Training data of low quality	Better quality of training data
The luminosity was low	Better lighting conditions
Lack of variety of images per class	Variety of training data
Unbalanced number of training data	Balanced training data
The pictures were taken from the same distance from the refrigerator	Several Angle and distances to the refrigerator when taking pictures

5.1.3 Chosen Parameter

Agarwal [1] concluded after comparing the model's accuracy before and after changing the configuration file parameters that the default configuration gives higher mAP values in all cases. Hence, we chose to use the default configuration of the Faster R-CNN model in our work. We only made some custom changes to the configuration to make the model fit our purposes. All the performance results of the Faster R-CNN model were visualized under the following updated configuration:

```
   The number of classes is four, used for our model
   Batch size: 1
   We kept the learning rate:  0.04
   We kept aspect_ratios as in default configuration: [0.5, 1.0, 2.0]
   Number of steps used is 25.000
   We updated the fine_tune_checkpoint to our path
   We set fine_tune_checkpoint_type to detection
   We updated the label_map_path to /home/bokhari/customod/TensorFlow/workspace/
  training_demo/annotations/label_map.pbtxt
   We set the input path input_path for the training to /home/bokhari/customod/
  TensorFlow/workspace/training_demo/annotations/train.record
   We set the input path input_path for the evaluation to /home/bokhari/customod/
  TensorFlow/workspace/training_demo/annotations/test.record
```

Listing 5.1: Modifications to the Configuration File

Previous studies in object detection inside refrigerators have primarily concentrated on evaluating the performance of various models and determining the optimal model for a specific dataset. In contrast, our research took a different approach by focusing on the dataset. We examined the significance of dataset quality and aimed to identify critical factors that impact the model's accuracy. Our study aimed to gain insights into the conditions and factors essential to enhance the performance of object detection inside fridges, ultimately leading to better optimization of the object detection process.

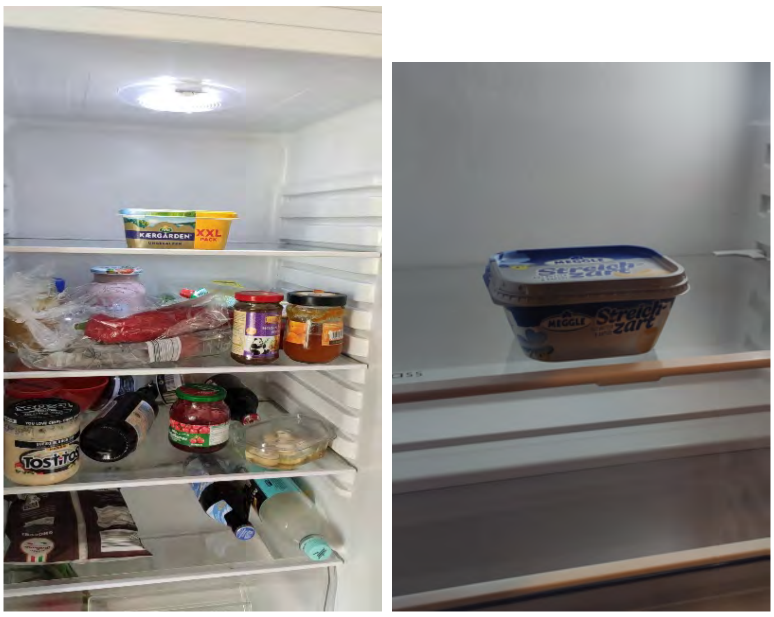


Figure 5.1.3: Comparison example between the two fridges

Figure 5.1.3 shows the difference in image quality between the two, which was critical in improving the model's accuracy.

5.2 Results from training the model with the first dataset and testing it with images from the first fridge

Tables 5.2.1 and 5.2.2 presents the results of the object detection model for detecting four objects of interest inside our fridge. By utilizing the first dataset to train the model and evaluating its performance using test images from the same dataset. The tables show the results of the model at various intersection over union (IOU) thresholds, after 25,000 training iterations.

Table 5.2.1 includes the mean average precision (mAP) metric, as well as the mAP

	mAP	mAP@.50IOU	mAP@.75IOU
Fine-tuned (25k)	0.2877	0.5408	0.2739

Table 5.2.1: Mean Average Precision values

at different intersection over union (IOU) thresholds, specifically mAP@.50IOU and mAP@.75IOU. These metrics measure the model's ability to detect the objects of interest within the fridge accurately. The mAP metric is a general indicator of the model's performance, while mAP@.50IOU and mAP@.75IOU specifically measure the model's performance at IOU thresholds of 50% and 75%, respectively. Unfortunately, the results are inadequate, with an mAP of only 28%. This indicates that the model is not performing well in detecting the objects of interest in the fridge.

	AR@1	AR@10	AR@100
Fine-tuned (25k)	0.2688	0.3688	0.3812

Table 5.2.2: Average Recall values

Table 5.2.2 includes the Average Recall (AR) metric at different levels of recall, specifically AR@1, AR@10 and AR@100. These metrics measure the model's ability to accurately detect the objects of interest within the fridge. The AR@1 metric reports the average recall when only considering the top detection, AR@10 reports the average recall when considering the top 10 detections and AR@100 reports the average recall when considering the top 100 detections. The results presented in the table indicate a relatively low value for the recall metric, suggesting that the object detection model is encountering difficulty in accurately identifying a substantial proportion of the objects of interest within the fridge.

Figure 5.2.1 shows how the detector could not detect the butter but could detect the carrot with 99% and 76% accuracy, even if it was not an image with a complicated background or with several objects. It shows that the model needed to be trained better using the images from the first dataset by adding more images for the carrot class to balance between the carrot and butter classes or by adding various images from different angles and positions.

Figure 5.2.2 shows that the model could only detect the butter class this time and not the carrot.

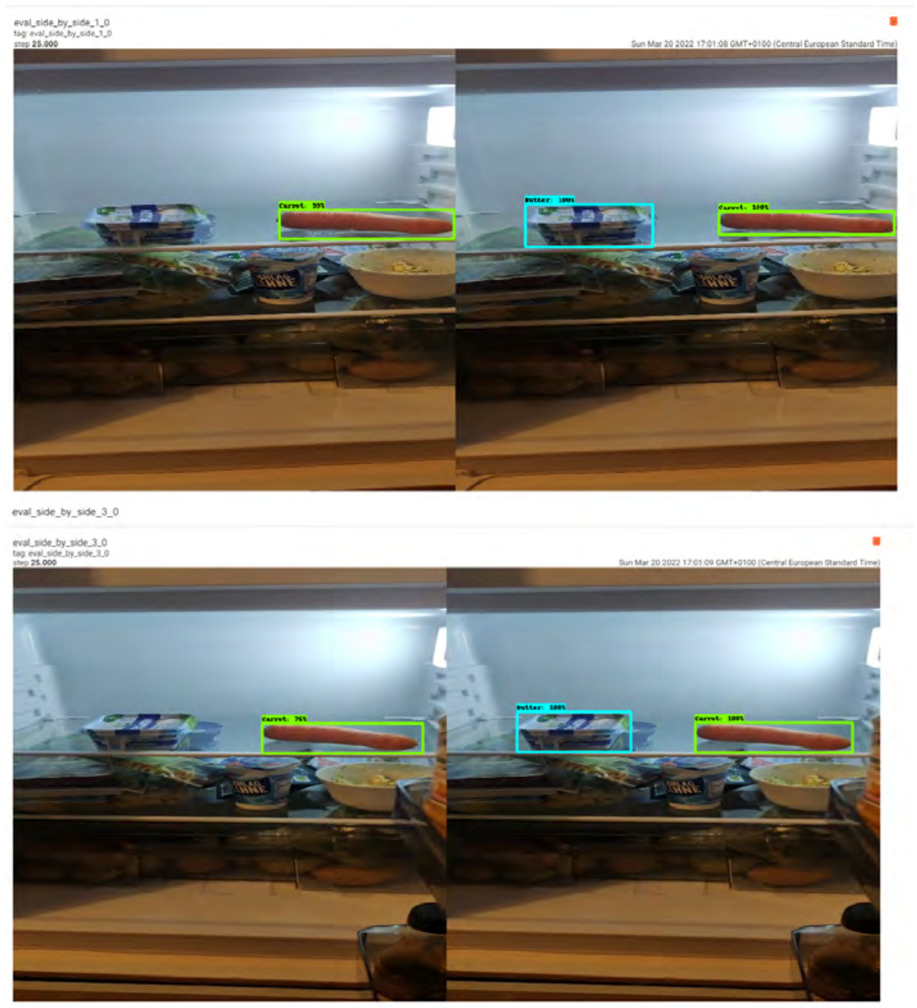

eval_side_by_side_1_0
tag: eval_side_by_side_1_0
step 25.000
Sun Mar 20 2022 17:01:08 GMT+0100 (Central European Standard Time)
Carrot: 99%
Butter: 100%
Carrot: 100%
eval_side_by_side_3_0
eval_side_by_side_3_0
tag: eval_side_by_side_3_0
step 25.000
Sun Mar 20 2022 17:01:09 GMT+0100 (Central European Standard Time)
Carrot: 76%
Butter: 100%
Carrot: 100%

Figure 5.2.1: Results show how the model could detect only the carrot with different accuracy values

Figure 5.2.3 shows how the model could not detect either of the two objects (carrot and butter), although the detector could detect the carrot, as shown in Figure 5.2.1 and the butter only in Figure 5.2.2.

5.2.1 Summary

The results show that the model could recognize the two objects separately, i.e., once it recognized the carrot class but not the butter class.

In some cases, it could recognize the butter but not the carrot, and sometimes it could recognize neither object. Several factors played a crucial role in the models recognition ability.

The quality of the training dataset, the shooting angle of the photos, the diversity of the dataset, the distance to the refrigerator, and the position of the photos could all have a

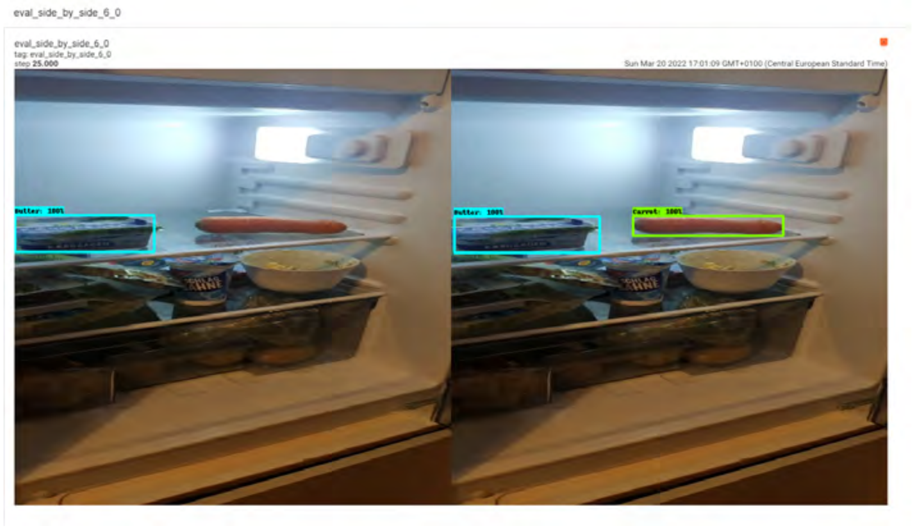

eval_side_by_side_6_0
eval_side_by_side_6_0
tag: eval_side_by_side_6_0
step 25.000
Sun Mar 20 2022 17:01:09 GMT+0100 (Central European Standard Time)
butter: 100%
butter: 100%
Carrot: 100%

Figure 5.2.2: The results show how the model was able to detect only the butter

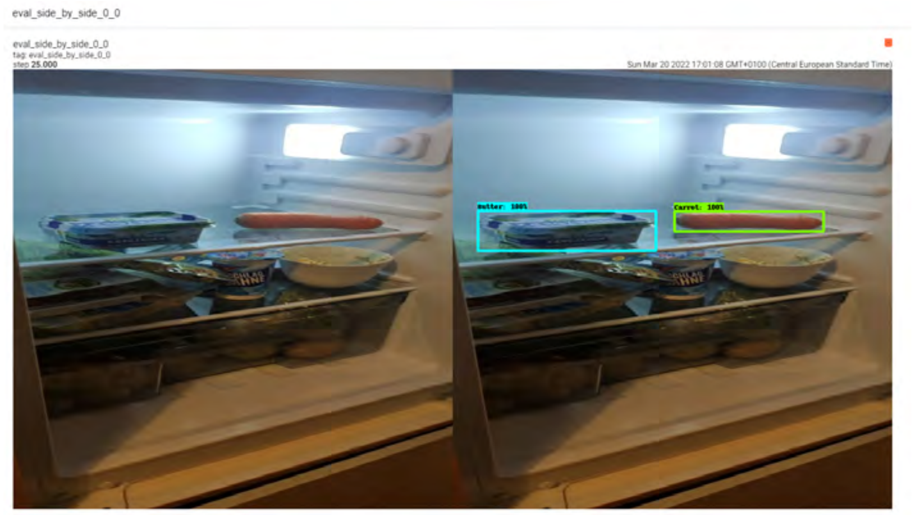

eval_side_by_side_0_0
eval_side_by_side_0_0
tag: eval_side_by_side_0_0
step 25.000
Sun Mar 20 2022 17:01:08 GMT+0100 (Central European Standard Time)
butter: 100%
Carrot: 100%

Figure 5.2.3: The results show how the model was unable to detect any object

significant impact on the models ability to generalize well and accurately recognize both classes simultaneously for different test examples.

5.3 Results from training the model with the first dataset and testing it with images from the second fridge

Tables 5.3.1 and 5.3.2 present the results of evaluating the object detection model's performance, obtained by training the model with the first dataset and subsequently testing it using images from a separate, second dataset.

	mAP	mAP@.50IOU	mAP@.75IOU
Fine-tuned (25k)	0.337	0.4591	0.3974

Table 5.3.1: Mean Average Precision values

Table 5.3.1 demonstrates the results of the object detection models performance, which has shown a slight improvement with a mean average precision (mAP) value of 33.7%. The improved results are attributed to the fact that the testing data of the second fridge was obtained under more favorable conditions, resulting in higher image quality. This, in turn, has led to more accurate detection of objects of interest within the fridge

	AR@1	AR@10	AR@100
Fine-tuned (25k)	0.3829	0.5147	0.5147

Table 5.3.2: Average Recall values

Table 5.3.2 illustrates that the values for the average recall metric have also exhibited an improvement.

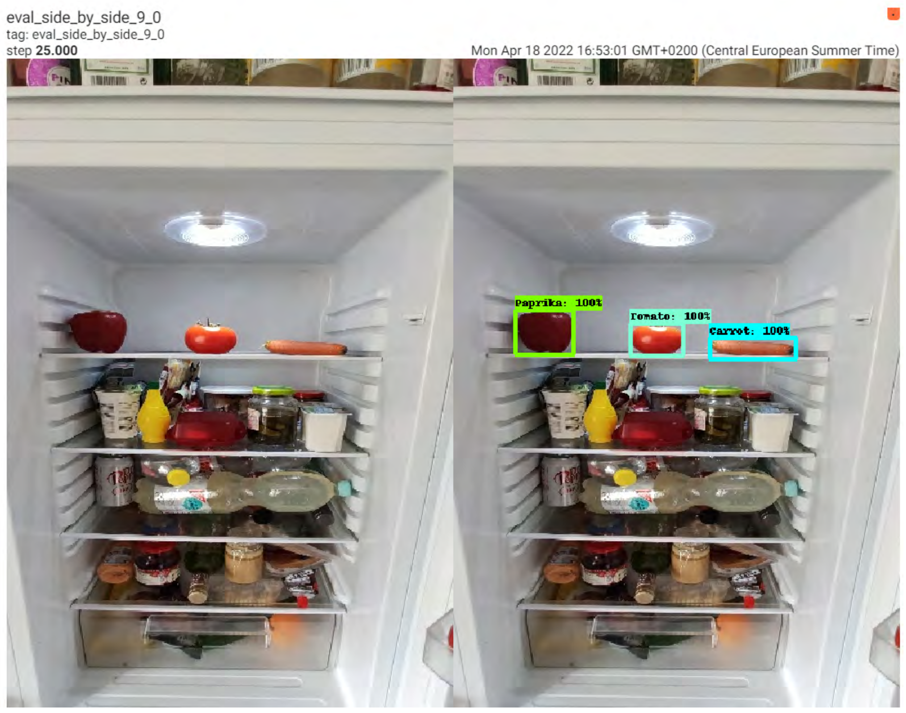

eval_side_by_side_9_0
tag: eval_side_by_side_9_0
step 25.000
Mon Apr 18 2022 16:53:01 GMT+0200 (Central European Summer Time)
Paprika: 100%
Tomato: 100%
Carrot: 100%

Figure 5.3.1: The model could not detect any of the target classes

As shown in Figure 5.3.1, the model could not detect any object classes in the new fridge after training with the first dataset with low-quality data. As a result, the model performed poorly in the test dataset, failing to detect any of the target classes.

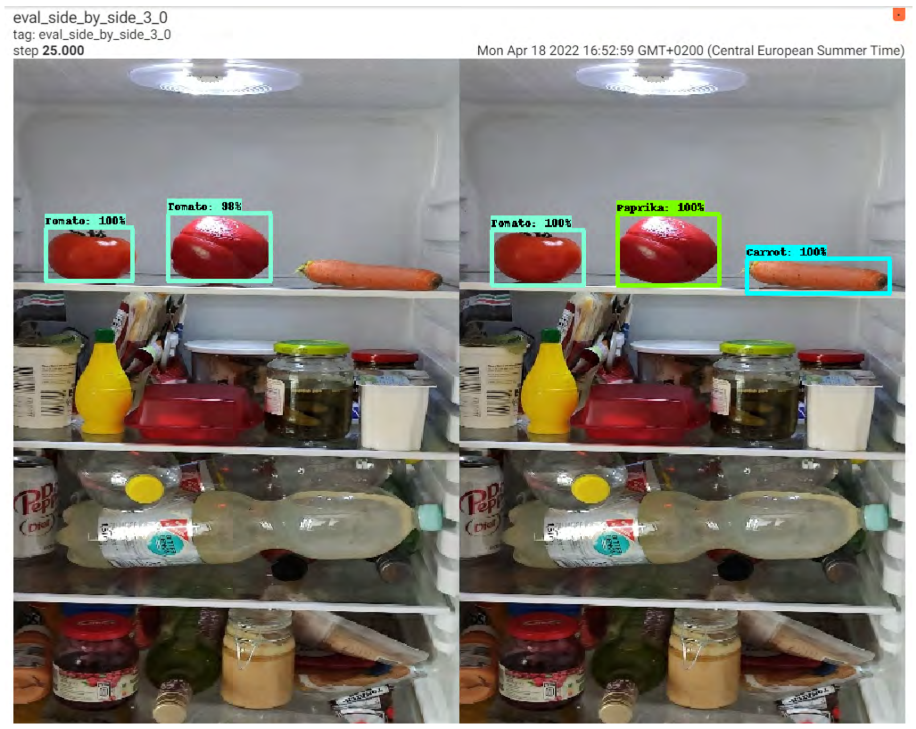


Figure 5.3.2: The model mixed up pepper and tomato

Figure 5.3.2 shows how the model falsely predicts the bell pepper as a tomato. This could be because of the similarity in shape and color since the red pepper had approximately the same shape as the tomato in the image and the same color. So, the model could not make a difference between the two classes. This problem could be addressed by taking more pictures of the red pepper class from different angles and positions to guarantee that the model learned more and overcame this issue.

Figure 5.3.3 shows two occluded objects, the bell pepper and tomato, with an additional carrot. The model could not detect the carrot and considered the red pepper and the tomato as one class (tomato).

Figure 5.3.4 contains three objects, the model mispredicted the bell pepper and considered as a tomato. The model couldn't detect the tomato and the carrot class

5.3.1 Summary

Training the model with the first dataset of low-quality images led to a poor job of detecting objects in the new fridge. The model failed to detect the carrot in most cases and repeatedly mistook the red pepper for a tomato, it also performed poorly in the occlusion problem.

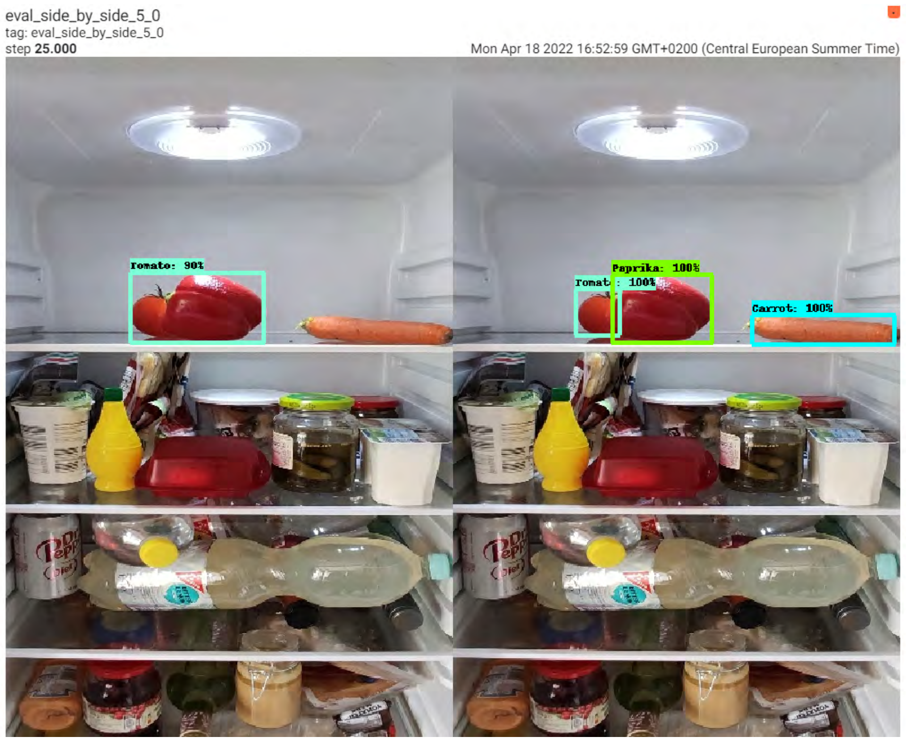

eval_side_by_side_5_0
tag: eval_side_by_side_5_0
step 25.000
Mon Apr 18 2022 16:52:59 GMT+0200 (Central European Summer Time)
Tomato: 90%
Paprika: 100%
Tomato: 100%
Carrot: 100%

Figure 5.3.3: The model did not do perform well in the occlusion problem

5.4 Results from training the model with 50 images pro class

Tables 5.4.1 and 5.4.2 demonstrate the outcome of assessing the model following the utilization of 50 images per class from the second dataset and evaluating it with images sourced from the same refrigerator.

	mAP	mAP@.50IOU	mAP@.75IOU
Fine-tuned (25k)	0.5219	0.7656	0.6558

Table 5.4.1: Performance Evaluation of the Model: mAP, mAP@.50IOU and mAP@.75IOU Results values

It is observed from Table 5.4.1 that the performance has improved in comparison to previous cases, and the mean average precision (mAP) has increased to 52%.

	AR@1	AR@10	AR@100
Fine-tuned (25k)	0.5004	0.7138	0.7163

Table 5.4.2: Performance Evaluation of the Model: Average Recall values

Table 5.4.2 indicates that the average recall values have increased, which implies that the

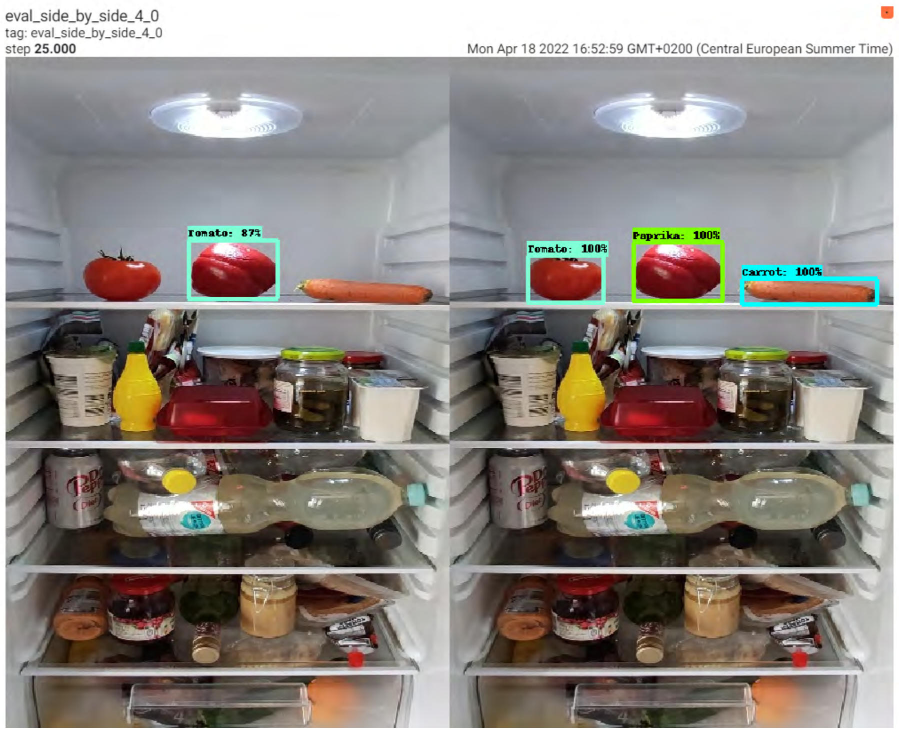

eval_side_by_side_4_0
tag: eval_side_by_side_4_0
step 25.000
Mon Apr 18 2022 16:52:59 GMT+0200 (Central European Summer Time)
Tomato: 87%
Tomato: 100%
Paprika: 100%
Carrot: 100%

Figure 5.3.4: The model could not detect the carrot and the tomato and falsely predicted the red pepper

model has shown enhancement in identifying the objects of interest.

Figure 5.4.1 shows that the model could detect the yellow pepper and the carrot but could not detect the tomato and the butter.

Figure 5.4.2 shows how the model mixed up the three peppers with the tomatoes, even though the peppers have different shapes and colors.

Figure 5.4.3 shows how the model correctly detected the butter class but misclassified the yellow and orange peppers as tomatoes.

Figure 5.4.4 shows how the model correctly detected the butter and carrot objects but confused the yellow pepper and tomato objects again.

We have changed the order of the objects and put the yellow pepper in the left position, as shown in Figure 5.4.5. The model did not detect the yellow pepper correctly and considered it again as a tomato, but he could detect the carrot and the butter classes correctly.

We finally put the yellow pepper in the middle, as shown in Figure 5.4.6. The model considered the yellow pepper like a tomato and detected the carrot and butter objects well.

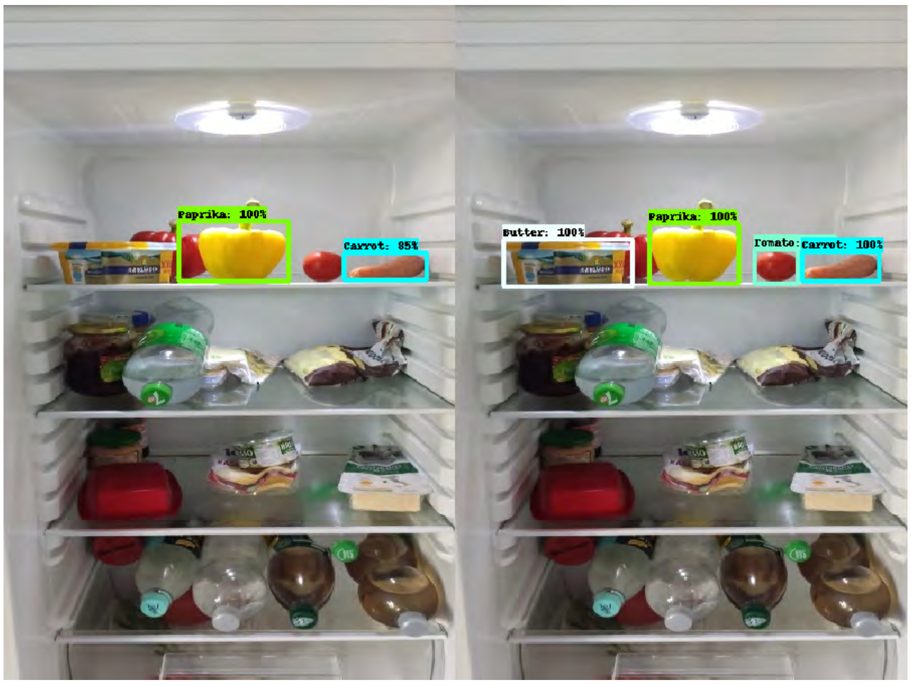


Figure 5.4.1: The result shows the model detecting only pepper and carrot

5.4.1 Summary

The detector performed admirably in detecting carrot, butter, and tomato objects, with only a few instances where the model failed to recognize the butter and tomato. However, it misidentified the bell pepper class and became confused between the tomato and the bell pepper.

5.5 Results from training the model with around 100 images pro class

Tables 5.5.1 and 5.5.2 present the outcome following the expansion of the training dataset from 50 images per class to 100, and the subsequent training of the model with the augmented dataset.

	mAP	mAP@.50IOU	mAP@.75IOU
Fine-tuned (25k)	0.6106	0.8606	0.6558

Table 5.5.1: Mean Average Precision values

Table 5.5.1 demonstrates that the expansion of the training dataset yields a notable enhancement in accuracy, as evidenced by an 11% increase in mAP.

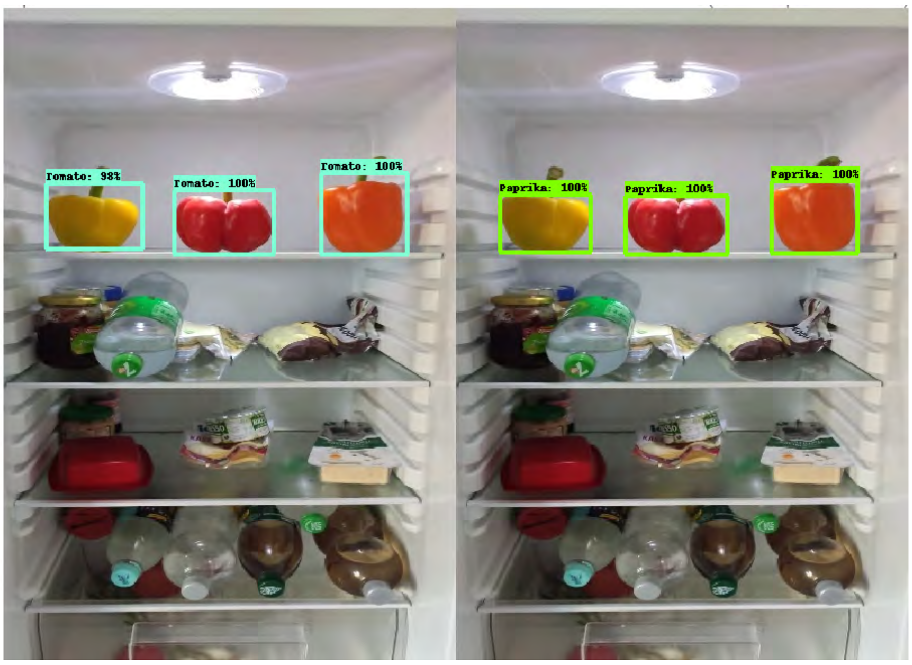


Figure 5.4.2: The result shows the model detecting the three peppers as tomatoes

	AR@1	AR@10	AR@100
Fine-tuned (25k)	0.6494	0.7244	0.7244

Table 5.5.2: Average Recall values

Table 5.5.2 illustrates that the expansion of the training dataset also results in an improvement in average recall values, indicating that the model now exhibits superior object detection capabilities as compared to its previous performance.

Figure 5.5.1 shows that the model could detect the three objects successfully. The model could detect the red bell pepper successfully. The model could overcome the problem of confusion between the bell pepper and tomato classes. This issue was addressed by increasing the number of images per class and providing more training images for the bell pepper class.

Figure 5.5.2 shows how the model did well in detecting the carrot, tomato, and one red pepper, but it could not detect the red bell pepper on the left side and considered it a tomato.

Figure 5.5.3 shows that the model could detect the three objects correctly. The model predicted the yellow bell pepper correctly this time and did not consider it a tomato like in the previous test.

We switched the positions between the butter and the carrot, as shown in Figure 5.5.4.

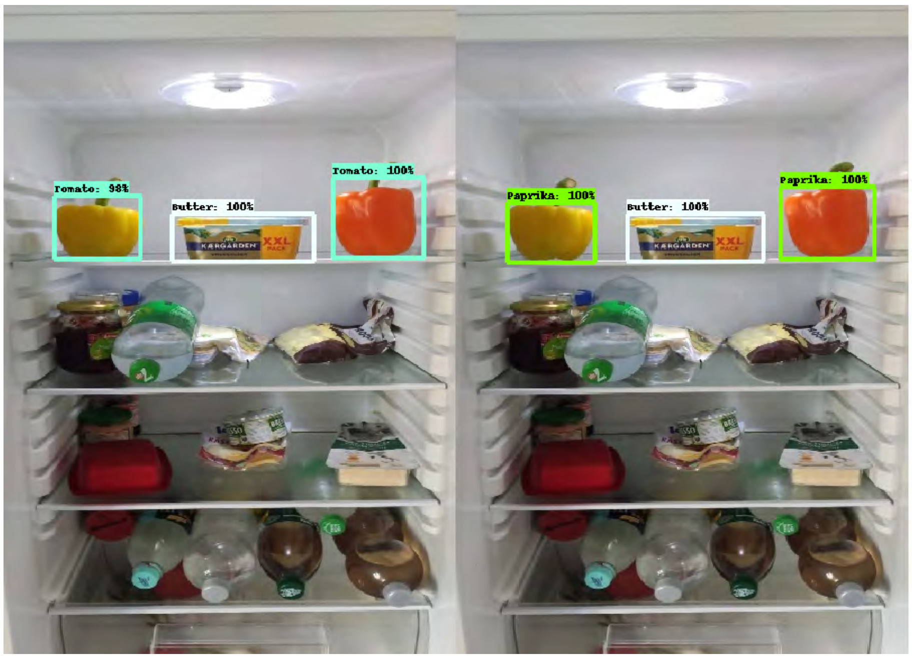


Figure 5.4.3: The result shows the model detecting the two peppers as tomatoes

The model did not get confused and detected the three objects correctly, which is a good sign of the model's ability to generate nicely.

This time, the model predicted the red bell pepper correctly. The previous tests confused the red bell pepper shown in the middle of Figure 5.5.5 with a tomato. The model detected and predicted the tomato and carrot classes correctly.

The model considered the red bell pepper and the tomato as one object class (in this case, the red bell pepper), which is the foreground class, as shown in Figure 5.5.6. However, the model could detect the carrot correctly.

The model treated the occluded objects, butter and tomato, as a single object and assigned them the label Butter class, as shown in Figure 5.5.7.

The model could detect the carrot correctly but could not detect the red bell pepper this time.

5.5.1 summary

Increasing the number of training images for each class improved detection performance and assisted the model in overcoming the tomato-bell pepper confusion problem.

The model could now do better at detecting the combined objects. The model took the foreground class as the predicted class in the case of an occlusion problem, as we saw in

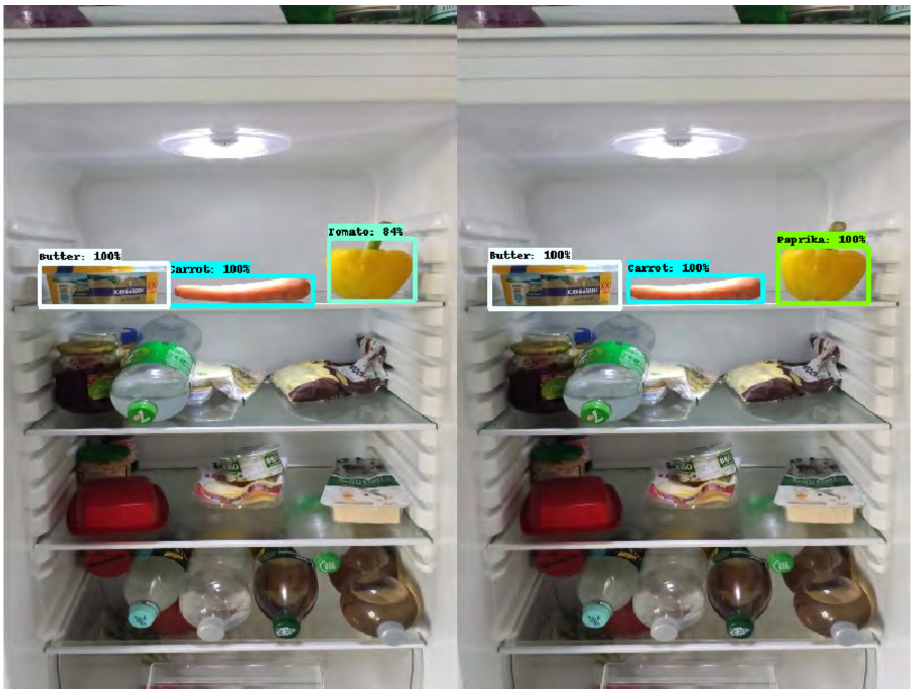


Figure 5.4.4: The result shows the model detecting the pepper as a tomato and correctly detecting the other classes

Figures 5.5.6 and 5.5.7.

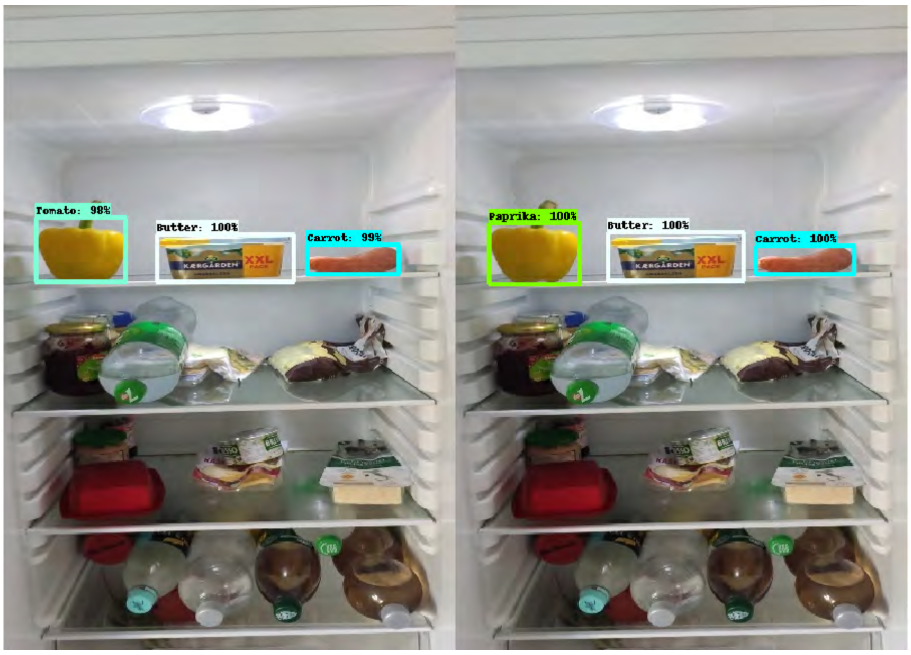


Figure 5.4.5: The result after changing the yellow pepper to the left position

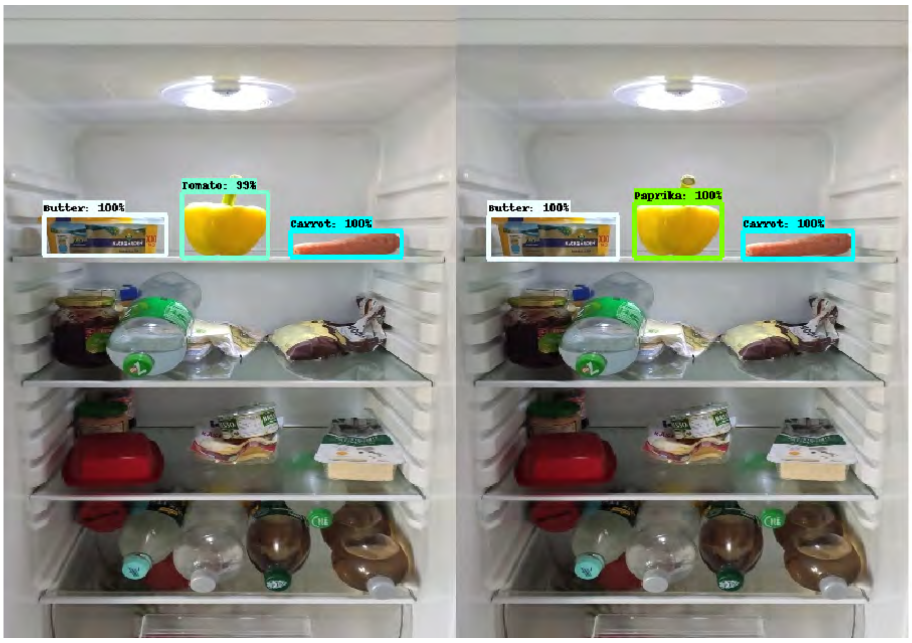
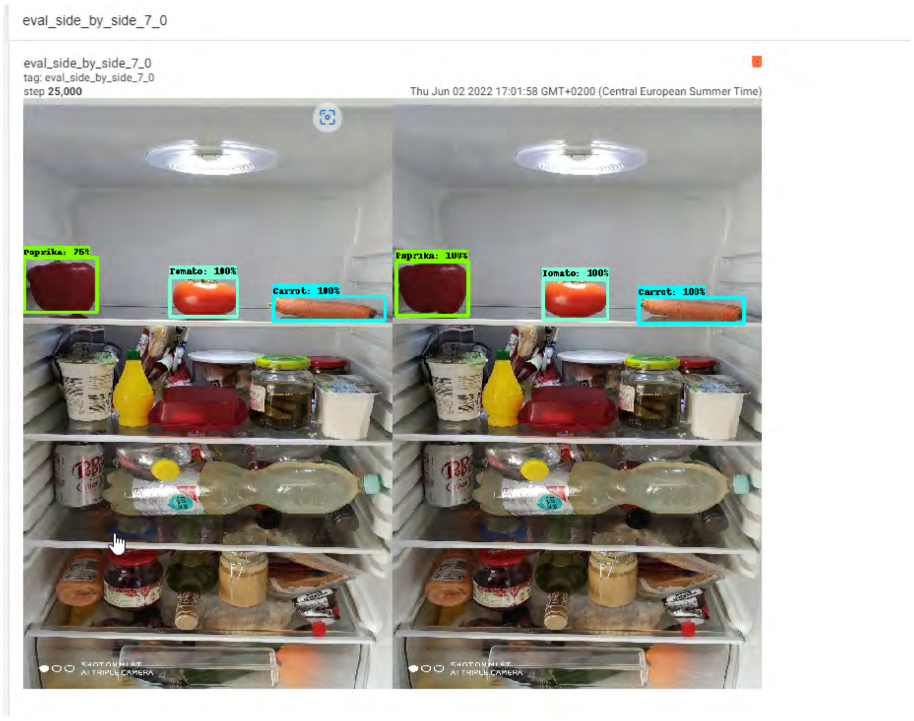
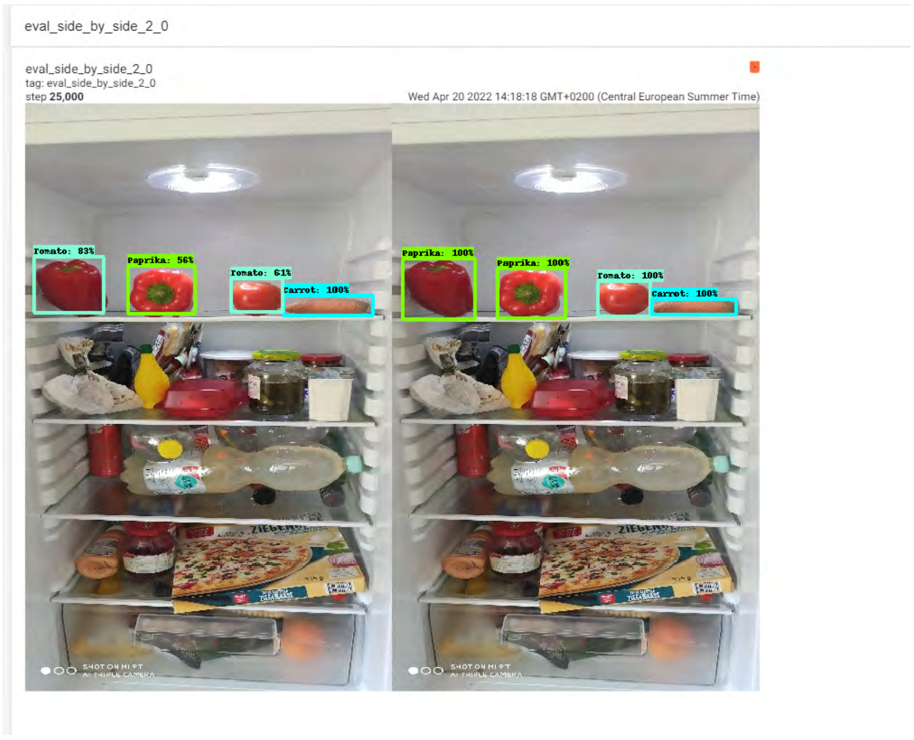


Figure 5.4.6: Putting the yellow pepper in the middle this time


eval_side_by_side_7_0

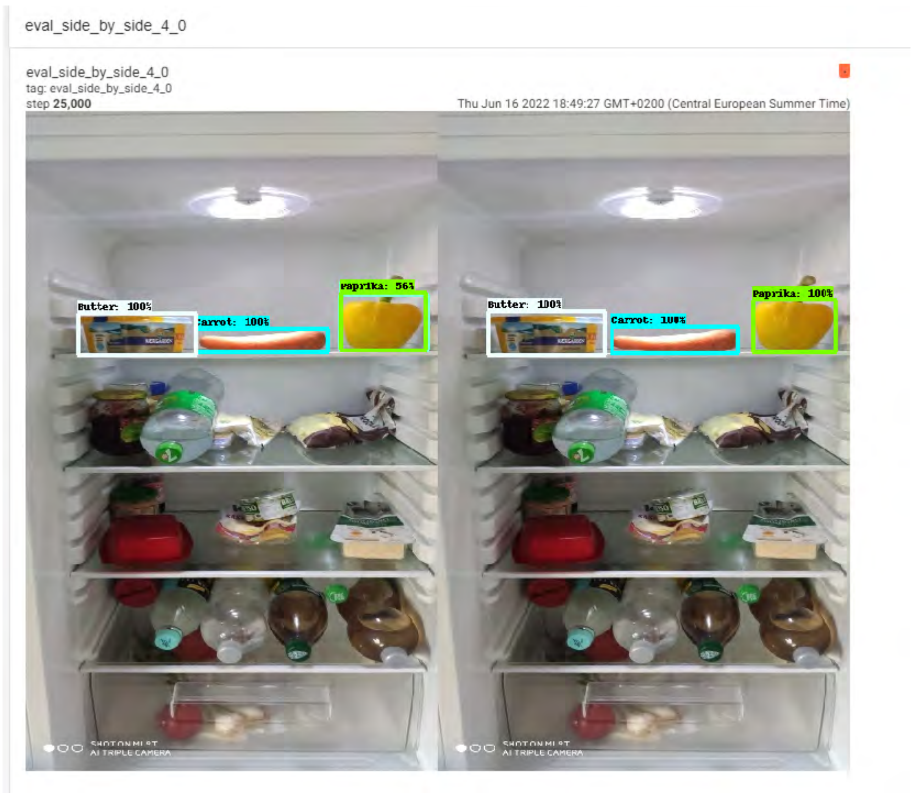
eval_side_by_side_7_0
tag: eval_side_by_side_7_0
step 25,000
Thu Jun 02 2022 17:01:58 GMT+0200 (Central European Summer Time)

Figure 5.5.1: The model detected successfully the three objects


eval_side_by_side_2_0

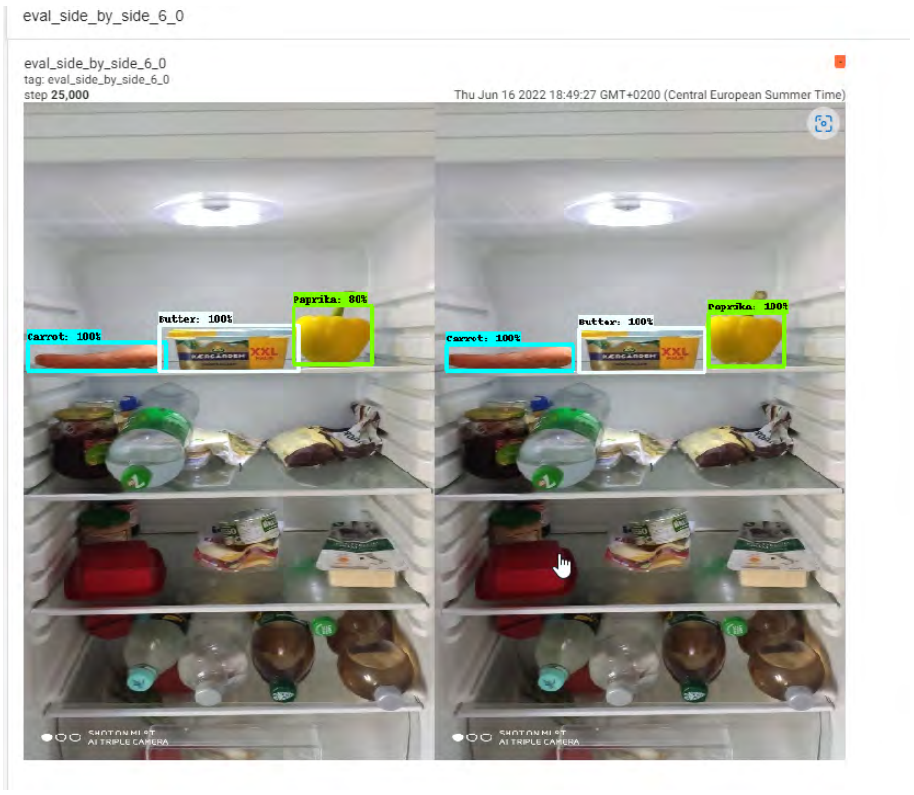
eval_side_by_side_2_0
tag: eval_side_by_side_2_0
step 25,000
Wed Apr 20 2022 14:18:18 GMT+0200 (Central European Summer Time)

Figure 5.5.2: The model confused the red pepper as tomato


eval_side_by_side_4_0

eval_side_by_side_4_0
tag: eval_side_by_side_4_0
step 25,000
Thu Jun 16 2022 18:49:27 GMT+0200 (Central European Summer Time)

Figure 5.5.3: The model made good predictions


eval_side_by_side_6_0

eval_side_by_side_6_0
tag: eval_side_by_side_6_0
step 25,000
Thu Jun 16 2022 18:49:27 GMT+0200 (Central European Summer Time)

Figure 5.5.4: The result after switching positions between the butter and carrot

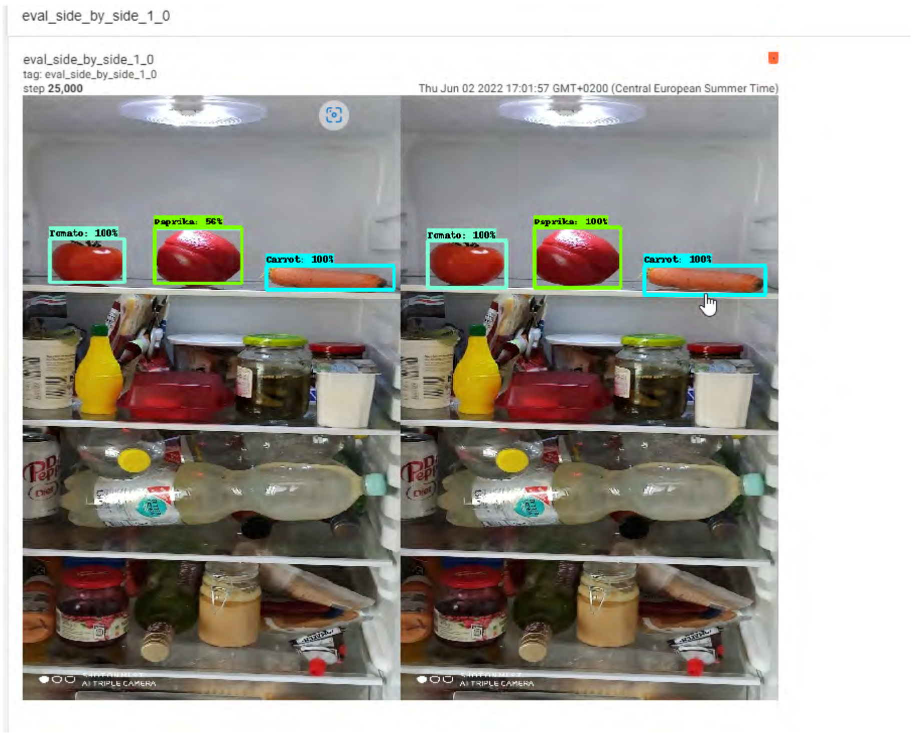


Figure 5.5.5: Some results example

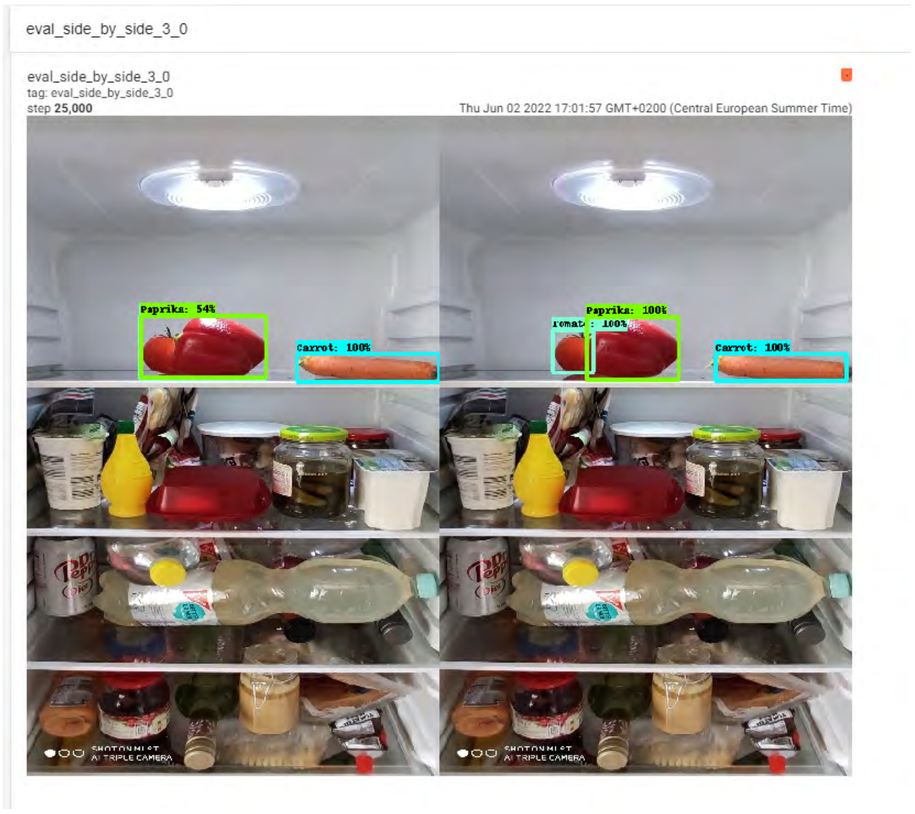
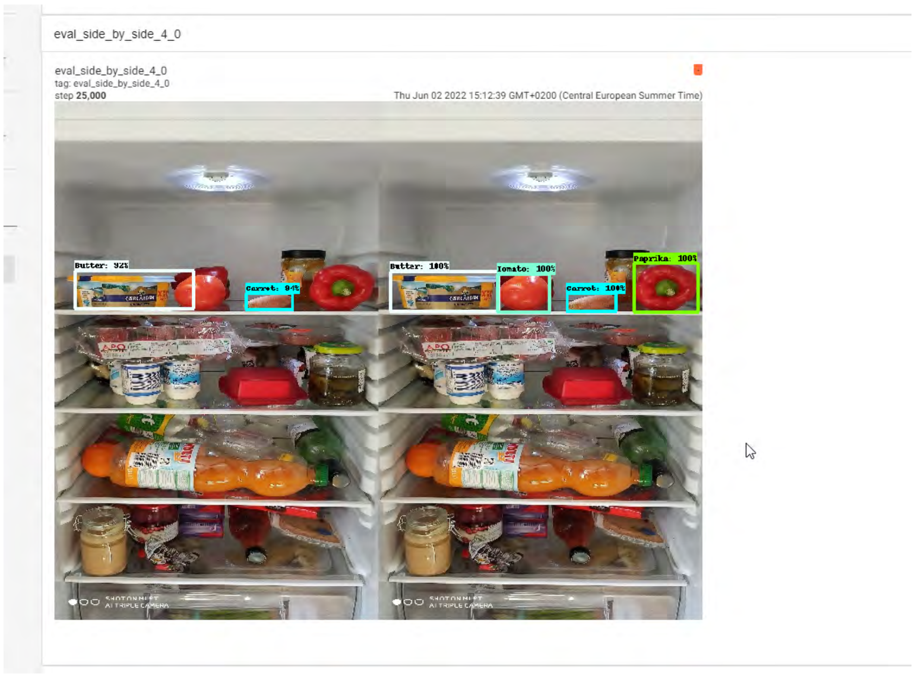


Figure 5.5.6: first example of occlusion


eval_side_by_side_4_0

eval_side_by_side_4_0
tag: eval_side_by_side_4_0
step 25,000

Thu Jun 02 2022 15:12:39 GMT+0200 (Central European Summer Time)

Figure 5.5.7: second example of occlusion

Chapter 6

Discussion

Table 6.0.1: A comparison of how well the model performed on the two datasets

Performance of the model in the first Dataset	Performance of the model in the second Dataset
The model performed poorly in recognizing object classes on the test data	The model performed well in recognizing object classes on the test data
The model's accuracy was poor.	The model's accuracy has improved.
The model needs to be improved on classes with similar shapes and colors.	The model could solve the issue of object confusion caused by similar shapes and colors.
The model was unable to detect the objects when they were combined.	The model performed well in detecting the objects when they were combined.

In this section, we discuss the performance results of the Faster R-CNN on the two datasets. In Table 6.0.1, we summarized the critical points of the differences in how the model performed while training it on the first and second datasets.

After reviewing several related papers, we chose the Faster R-CNN model for our study since it successfully detected contents inside the fridge. It was pre-trained on the COCO 2017 dataset. Thus, choosing a suitable model for a specific problem could be critical, as not all models perform well on a specific issue.

After doing experimental research on fine-tuning and training the model on two different datasets, we found that several factors could be crucial in affecting the model's performance.

We collected the first dataset from a small fridge with poor lighting and did not take photos of the objects from various angles and positions. We also ignored the fact that the distance from the fridge, no matter how far or close, should affect the detector's performance. As we can see in the results section, the model struggled to detect the objects when they were combined and became confused several times between the bell pepper and the tomato. This confusion was caused by the tomato's and bell pepper's similar shapes and colors. The model predicted the bell pepper as a tomato because the dataset was also unbalanced, as we had more images of a tomato in training than a bell pepper. Hence, the model has always considered the bell pepper a tomato.

We conclude from analyzing the first dataset that the dataset's quality and variety play a vital role in improving the detection ability of the object detector.

We have improved the quality of the second dataset by taking photos from a bigger, brighter fridge in another kitchen. We also considered that the dataset varied and was balanced, so we conducted an experiment in which we took 50 photos for each class, and the results were better than in the first refrigerator. We trained the model with photos taken from different angles and positions. We have improved the luminosity, thus improving the quality of the dataset. We took photos from various distances away and close to the fridge to help the model in the test dataset generalize better when detecting the combined objects. This led to better results, but the number of training images for each class needed to be increased to ensure better model performance. Because The model sometimes stuck between the bell pepper and the tomato and could not detect some objects.

We then took around 100 images per class, which led to good results and allowed the model to overcome the confusion between the objects. Furthermore, the object detector was capable of detecting combined objects reasonably well. Only in a few cases did the object detector predict incorrectly, but expanding the dataset would improve detection even more.

In our experimental research, we faced the problem of occlusion between objects. We noticed from the results that the model predicted the foreground class as the dominant class and considered the occluded objects as one object. The model could not detect the object in the background, and due to the limitations of our work, we did not address the occlusion issue.

Summary

We concluded that the following factors could play an essential role in improving the detection performance of an object detector:

- The dataset quality
- Light conditions
- Variety of training data, including shooting from different angles and positions, from different distances to the fridge
- Balanced training data

The Faster R-CNN model did pretty well in detecting the target objects, and the better the quality and variety of the dataset, the better performance of the object detector in detecting the target objects. Increasing the number of images per class leads to better results. The optimal number of training images per class would be around 200 to 250 images class to achieve perfect results. However, in the limitation of our work, we trained our model with around 100 images per class, which gave good results in detecting four objects inside our refrigerator. Another limitation of our work is that the model considers the two occluded objects as one. The front object class is considered the predicted class for the two objects, and the model could not recognize the hidden object on the backside.

Chapter 7

Conclusions

In conclusion, this thesis has presented a novel approach to content detection inside fridges using deep learning techniques. We focused on investigating the factors that play an important role in content recognition in the refrigerator. We limited our research to the detection of four target objects on the top shelf of our refrigerator. The proposed method utilizes a Faster R-CNN model pre-trained on COCO 2017 dataset and then trained on a dataset of images collected from a real-world fridge environment. The results showed that the model achieved an mAP of 62% in detecting these objects, which could be improved by adding more training data. We compared two datasets and analyzed several factors that could affect an object detector's performance. After several experiments, we found that the following factors in making a dataset played a crucial role in affecting the accuracy of the model:

- The brightness
- The balance of the dataset
- The angle of capture
- The distance between the objects
- The diversity of the dataset
- The quality of the images
- The similarity between the objects in terms of their colours and shapes
- The distance of taking pictures from the refrigerator

One of the significant contributions of this thesis is the creation of a dataset of fridge images that includes many objects inside the fridge under different lighting conditions and backgrounds. This dataset can be used as a benchmark for future research in this area and will help advance the field of object detection in a refrigerator environment.

The suggested system could be expanded and used in another context, such as tracking food stock or detecting expired items, thus simplifying the process of fridge management. Additionally, integrating the object detection system into smart kitchen appliances enhances the user experience, provides additional functionality, and can potentially decrease the cost of these devices as currently, smart fridges are quite costly.

The current system does have certain limitations, one of which is that the model has only been trained on four specific objects. We discovered that when two objects are occluded, the model incorrectly identifies the object in the foreground as the class predicted by the model, as we did not account for the occlusion issue in our research. Therefore, the model is currently unable to handle the occlusion issue. Future research could examine the usage of other deep learning models, such as Single Shot Detector (SSD) or YOLO, and expand the number of objects of interest. In future studies, addressing and developing solutions for the occlusion problem would be a valuable addition to this work.

This thesis has presented a comprehensive approach to object detection inside a fridge using deep learning techniques. The proposed system achieved pretty good accuracy in detecting four objects of interest and has the potential to be applied in various scenarios. The dataset created for this research can also serve as a benchmark for future studies. This work demonstrates the potential of object detection in the context of smart kitchen appliances and highlights the importance of further research in this field.

Bibliography

[1] Agarwal, Kirti. “Object detection in refrigerators using Tensorflow”. PhD thesis. 2018.

[2] Alpaydin, Ethem. Introduction to machine learning. MIT press, 2020.

[3] Ashwathan, R, Asnath, Victy Phamila Y, Geetha, S, and Kalaivani, K. “Object Detection in IoT-Based Smart Refrigerators Using CNN”. In: The Industrial Internet of Things (IIoT) Intelligent Analytics for Predictive Maintenance (2022), pp. 281–300.

[4] Al-Azzoa, Fadwa, Taqia, Arwa Mohammed, and Milanovab, Mariofanna. “Human related-health actions detection using Android Camera based on TensorFlow Object Detection API”. In: International Journal of Advanced Computer Science and Applications 9.10 (2018).

[5] Bharati, Puja and Pramanik, Ankita. “Deep learning techniquesR-CNN to mask R-CNN: a survey”. In: Computational Intelligence in Pattern Recognition (2020), pp. 657–668.

[6] Chen, Zhong, Zhang, Ting, and Ouyang, Chao. “End-to-end airplane detection using transfer learning in remote sensing images”. In: Remote Sensing 10.1 (2018), p. 139.

[7] Gao, Xiaoyan, Ding, Xiangqian, Hou, Ruichun, and Tao, Ye. “Research on food recognition of smart refrigerator based on SSD target detection algorithm”. In: Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science. 2019, pp. 303–308.

[8] Girshick, Ross. “Fast R-CNN”. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV). Dec. 2015.

[9] Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2014, pp. 580–587.

[10] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. “Imagenet classification with deep convolutional neural networks”. In: Communications of the ACM 60.6 (2017), pp. 84–90.

[11] Krogh, Anders. “What are artificial neural networks?” In: Nature biotechnology 26.2 (2008), pp. 195–197.

[12] Lee, Tae-Ho, Kang, Shin-Woo, Kim, Taehyun, Kim, Jin-Sung, and Lee, Hyuk-Jae. “Smart Refrigerator Inventory Management Using Convolutional Neural Networks”. In: 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE. 2021, pp. 1–4.

[13] Lin, Tsung-Yi, Maire, Michael, Belongie, Serge, Hays, James, Perona, Pietro, Ramanan, Deva, Dollár, Piotr, and Zitnick, C Lawrence. “Microsoft coco: Common objects in context”. In: European conference on computer vision. Springer. 2014, pp. 740–755.

[14] Padilla, Rafael, Netto, Sergio L, and Da Silva, Eduardo AB. “A survey on performance metrics for object-detection algorithms”. In: 2020 international conference on systems, signals and image processing (IWSSIP). IEEE. 2020, pp. 237–242.

[15] Pan, Sinno Jialin and Yang, Qiang. “A survey on transfer learning”. In: IEEE Transactions on knowledge and data engineering 22.10 (2009), pp. 1345–1359.

[16] Rajpura, Param S, Bojinov, Hristo, and Hegde, Ravi S. “Object detection using deep cnns trained on synthetic images”. In: arXiv preprint arXiv:1706.06782 (2017).

[17] Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: Advances in neural information processing systems 28 (2015).

[18] Uijlings, Jasper RR, Van De Sande, Koen EA, Gevers, Theo, and Smeulders, Arnold WM. “Selective search for object recognition”. In: International journal of computer vision 104.2 (2013), pp. 154–171.

[19] Won, Wei-Cheng, Yong, Yoke-Leng, and Khor, Kok-Chin. “Object Detection and Recognition for Visually Impaired Users: A Transfer Learning Approach”. In: 2021 2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS). IEEE. 2021, pp. 1–6.

[20] Wu, Peng, Sun, Bei, Su, Shaojing, Wei, Junyu, Zhao, Jinhui, and Wen, Xudong. “Automatic modulation classification based on deep learning for software-defined radio”. In: Mathematical Problems in Engineering 2020 (2020).

[21] Zaniolo, Luiz and Marques, Oge. “On the use of variable stride in convolutional neural networks”. In: Multimedia Tools and Applications 79.19 (2020), pp. 13581–13598.

[22] Zhou, Zhi-Hua. Machine learning. Springer Nature, 2021.