Four Eyes see more than two: Machine Learning-based Object Detection in Augmented Reality

Masterarbeit

Jan Niklas Engel | 1730641
Wirtschaftsinformatik

TECHNISCHE UNIVERSITÄT DARMSTADT

WIRTSCHAFTS INFORMATIK

UNIVERSITY OF CAMBRIDGE

CYBER HUMAN LAB

Jan Niklas Engel
Matrikelnummer: 1730641
Studiengang: M.Sc. Wirtschaftsinformatik

Masterarbeit
Thema: Four Eyes see more than two: Machine Learning-based Object Detection in Augmented Reality.

Eingereicht: 24. Juli 2021

Betreuer: M.Sc. Luisa Pumplun

Prof. Dr. Peter Buxmann
Fachgebiet Wirtschaftsinformatik | Software & Digital Business
Fachbereich Rechts- und Wirtschaftswissenschaften
Technische Universität Darmstadt
Hochschulstraße 1
64289 Darmstadt

Betreuer: Dr. Thomas Bohné

Cyber-Human Lab | Institute for Manufacturing
Department of Engineering
University of Cambridge
17 Charles Babbage Road
CB3 0FS Cambridge, United Kingdom

Ehrenwörtliche Erklärung gemäß § 22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jan Niklas Engel, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Darmstadt, den 24. Juli 2021

Abstract

First strides have been made to successfully integrate Machine Learning based quality inspection into production lines. Nevertheless, many processes still require human sampling testing and cannot be fully automated. In these hybrid inspection scenarios digital transformation is not about automation but rather about assisting and increasing human performance. One approach to do so could be to improve visual quality inspection by integrating Machine Learning models into Augmented Reality applications. This thesis describes the development of a Machine Learning based object detection system for the HoloLens. Instead of off-loading the inference step to external resources, the system uses the Barracuda inference engine to execute a Tiny YOLOv2 object detection model on the device. The evaluation of the object detection model on the PASCAL VOC benchmark shows that systems only using technological frameworks which are fit for AR can achieve the same accuracy as Machine Learning models that are backed up by established frameworks. With its modular design, this proof of concept implementation serves as a blueprint for future applications and is a first step towards the successful integration of Machine Learning based Augmented Reality systems into industrial quality inspection settings.

Table of Contents

List of Figures

List of Tables

Nomenclature

API	Application Programming Interface
AR	Augmented Reality
AV	Augmented Virtuality
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
DSR	Design Science Research
DSRM	Design Science Research Methodology
fps	Frames per Second
GPU	Graphics Processing Unit
HMD	Head-mounted Display
IMGUI	Intermediate Mode Graphical User Interface
IMU	Inertial Measurement Unit
IL2CPP	Intermediate Language To C++
IoU	Intersection over Union
mAP	Mean Average Precision
ML	Machine Learning
MP	Megapixel
NHWC	Tensor Format: Number of samples, Height, Width, Channels
NMS	Non-Max Suppression
N/A	Not Available
ONNX	Open Neural Network Exchange
PV	Photo/Video
R-CNN	Region-based Convolutional Neural Network
RGB	Image Format: Red, Green, Blue
uGUI	Unity Graphical User Interface
UWP	Universal Windows Platform
VOC	Visual Object Classes
WinML	Windows Machine Learning
WLAN	Wireless Local Area Network
YOLO	You Only Look Once

1 Introduction

1.1 Motivation

Suppose a baker who has been making the same sort of bread for many years. He has perfected his art and makes sure he always uses the same ingredients so that all breads have the same shape, weigh the same, and – most importantly – taste equally delicious. Occasionally though, lost in thought in the early morning, the baker makes a mistake and adds too much salt to the dough, leaving his customers unhappy at their breakfast table.

Although we do not want to concern ourselves with the baking business, this practical example highlights an important and recurring problem: industries that rely on manual labour have an inherent potential for human error. No matter how thorough someone performs their duty or how well trained or experienced a worker is, if humans are involved in a process, there is room for mistakes.[1] One of the industrial tasks, for which human expertise is still highly valuable, is quality inspection. However, operators' inspection performance is easily affected by personal factors such physical and cognitive condition. Not only this, but the pace and complexity of the task[2] as well as the inspection method and strategy[3] can also influence the performance and make the inspection process prone to errors. When going undetected, these errors can cost become very costly for the company. Defective goods have to be replaced and discontent among customers might lead to lower sales in the future.

One way to reduce the risk for errors in quality inspection is automation.[4] As a result of continued digital transformation, an increasing number of tasks can be automated, thereby substituting human work. First strides have been made to successfully integrate Machine Learning (ML)-based quality inspection into production lines, augmenting the inspection process and e.g. reducing scrap trough early control interventions.[5] Nevertheless, many processes still require human sampling testing and cannot be fully automated. This makes these processes a case for hybrid inspection approaches, where "digital transformation is not

[1] Cf. Mital et al. (1998).

[2] Cf. Eskew; Riche (1982), Gallwey; Drury (1986).

[3] Cf. Su; Konz (1981), Drury et al. (1983).

[4] Cf. Kopardekar et al. (1993).

[5] Cf. Schmitt et al. (2020).

about automation but rather about assisting and increasing human performance".[6] Therefore, it seems promising to explore a hybrid inspection approach, combining human and machine capabilities by providing operators with augmented inspection assistance.

1.2 Objective and Outline

One approach would be to improve visual quality inspection by integrating ML models into AR applications in order to get a second opinion on the spot. ML models have shown great results for a multitude of visual tasks.[7] In addition, head-mounted displays (HMDs) have the sensors to provide these models with the necessary data and the capability to visualise the results in an AR. At the same time, they allow operators to use their hands without being impaired by a manually operable interface.[8]

The integration of a ML Model into an HMD, however, poses some additional challenges. There is the limited computing capacity of such devices and the lack of support for well-established ML frameworks like TensorFlow or PyTorch. Given these constraints, how can such a system be successfully implemented? In trying to answer this question, the main part of this thesis considers the implementation of an AR application that uses a ML model. In doing so, the aforementioned challenges and possible solutions are discussed. The system should run exclusively on the HMD and be able to detect common objects. After implementation and evaluation, the application is supposed to serve as a blueprint artefact for other visual ML systems. Adapting it to specific visual quality inspection tasks is then merely a question of training the appropriate ML models, an issue which has already been given a lot of attention.[9]

The availability of a solution that builds on the interaction between AR and ML will make it easier for companies to understand the state of the technologies as well as the benefits they could bring to their processes. Enhancing the reciprocal relationship between worker and technology will smoothen the transition into industry 4.0 by providing a more collaborative approach of automation. A system that reliably uses ML for object detection in industrial AR devices would build the basis to not only improve human performance on visual quality inspection but could also help for a variety of other problems. Future research can be

[6] Cf. Krenzer et al. (2019), pp. 1f.

[7] Cf. Wu et al. (2019).

[8] Cf. Angrisani et al. (2020).

[9] Cf. Sinha et al. (2017).

dedicated to the transfer of the system into other problem domains and the enablement of AI in settings where it has not been applicable so far.

The remainder of this thesis is structured according to the Design Science Research (DSR) publication.[10] Section 2 investigates state of the art research in AR and ML-based object detection. A rigorous study of recent literature for applications that combine both technologies is conducted and a framework for the classification of such artefacts derived. This is joined by a summary of the precise research gap and the resulting research question. In Section 3 the underlying research approach, design choices and methods for the evaluation are presented. Section 4 describes the design and architecture of the artefact. Section 5 discusses the technical requirement, followed by a description of the implementation in Section 6. In the next section, the artefact is evaluated in terms of validity and utility and the evaluation results are presented. This is followed by a discussion in Section 8, while the subsequent section concludes the thesis with summary.

[10] Cf. Gregor; Hevner (2013), pp. 349-351.

2 Related Work

This section aims at providing all the related research that is relevant to the problem at hand. Important terms are being defined and introduced (Section 2.1) and related contributions are presented to underline the research gap (Section 2.2). Additional knowledge for the construction of the system is presented in the form of state-of-the-art object detection models (Section 2.3) and finally the research gap and the research questions (RQ) are elaborated (Section 2.4).

2.1 Augmented Reality

The term Augmented Reality (AR) describes a situation, where the virtual world blends into the real world. We speak of AR as soon as virtual information, such as holograms, is positioned on top of the physical environment that is being observed.[11] In contrast to augmenting the real world with virtual objects, Augmented Virtuality (AV) is about integrating real-world elements into the virtual world – like interaction with virtual objects.[12] Because these distinctions have not been made exceedingly clear for some time, Milgram et al (1994) introduced the Reality-Virtuality (RV) Continuum (cf. Figure 1). This taxonomy gives credit to the fact that there is a broad range of stages between a real world and a solely virtual environment, all summarised under the term Mixed Reality (MR). Although other taxonomies have been proposed, the RV continuum is sufficient to follow the scope of this thesis.[13]

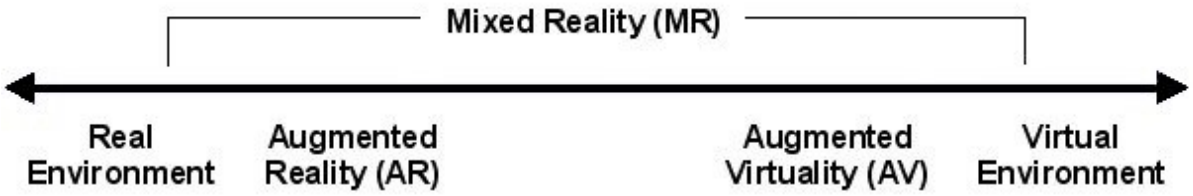


Figure 1: Reality-Virtuality Continuum by Milgram et al (1994).

[11] Cf. Szajna et al. (2019).

[12] Cf. Bruder et al. (2010).

[13] Cf. Milgram; Kishino (1994), Edwards-Steward et al. (2016).

2.1.1 Applications

Today the digitisation of production environments in the course of the fourth industrial revolution (Industry 4.0) makes AR one of the most interesting technologies for both research and industry in various domains.[14] There are many applications for higher education in science, technology, engineering, and mathematics[15] as well as for workforce training[16]. Del Amo et al. (2018) use AR to improve flexibility and adaptability of workers through augmented information perception in maintenance, while Palmarini et al. (2018) provide a review of how the technology is being employed in the aviation industry, plant maintenance, mechanical maintenance, consumer technology, the nuclear industry, and for remote (expert) systems. It also plays a role in health-care as a means to improve surgery[17] and is of growing importance in industrial quality inspection, e.g. by using brain-computer interfaces[18], as well as for infrastructure inspection[19]. A lot of research effort has been put into the application of AR technology in manufacturing[20], disassembly for remanufacturing[21], and assembly itself[22]. One example is the work of Henderson and Feiner (2011), who have shown that using AR can reduce time and errors by as much as 50% for manufacturing and assembly operations.

2.1.2 Head-Mounted Displays

In industrial contexts, the use of HMDs presents great potential because HMDs do not obstruct freedom of movement for hands. Interest has grown ever more since Microsoft made the technology available to a wider audience of researchers and developers by introducing the HoloLens 1 back in 2016.[23] Even though the technology behind HMDs is still under development and limitations exist with regard to input, field of view, tracking, and occlusion, their sensors provide more than a mere 2D visual perception of the world.[24] Modern devices

[14] Cf. Szajna et al. (2020).
[15] Cf. Ibáñez; Delgado-Kloos (2018).
[16] Cf. Damiani et al. (2018).
[17] Cf. Vávra et al. (2017).
[18] Cf. Angrisani et al. (2020).
[19] Cf. Mascareñas et al. (2020).
[20] Cf. Govindarajan et al. (2018), Yew et al. (2016), Egger; Masood (2020).
[21] Cf. Chang et al. (2017).
[22] Cf. Evans et al. (2017).
[23] Cf Dhiman et al. (2018).
[24] Cf. Miller et al. (2020).

are equipped with high resolution RGB video cameras, depth sensors for 3D scanning and distance measurement via time of flight techniques, as well as inertial measurement units (IMUs) that provide awareness of body movement. The newer Microsoft HoloLens 2 (2019), mostly targeted at Industrial AR applications, can gain an additional understanding of its environment via world-scale positional tracking (6 degrees of freedom) and spatial mapping. [25] With HMDs generating large amounts of data, they make a very interesting case for data intensive ML applications. Roth et al. (2020), for example, propose an approach that uses IMU data to recognise human activities with classification through ML techniques.

Both, the wide range of industrial AR applications and the growing capabilities of HMDs with their sensors, support the notion that performing more research more research on systems that make joint use of the capabilities of AR and ML presents a promising approach.

2.2 Machine Learning-based Augmented Reality

In order to identify applications that use both ML and AR a literature review was conducted by investigating recent papers that propose such systems. Overall, 15 papers were considered relevant, subsequently analysed and categorised according to the criteria in Table 1. These criteria have been derived based on the analysis. It became evident that at the moment, the main question is not so much in which domain these systems are deployed, but rather how the underlying technologies are combined. To answer this question, criteria one and two describe the immediate purpose of the integration of AR (“AR task”) and ML (“ML task”) respectively. Criterion three determines on which AR device the system is deployed (“AR Hardware”), thus taking into account the different maturity levels and platform dependencies of such devices. Given the high computational cost, criterion four ultimately determines whether the ML model is run on the same device as the AR task or is offloaded to another infrastructure ("ML Hardware"). Even though the literature review was not exhaustive and leaves room for bias, it shows that there is a tendency for the ML task to focus on object detection. The majority of authors researched object detection tasks with the execution of ML models equally divided between on- and off-device computations. While AR is primarily used for visualisation, the device usage is balanced between mobile and HMDs. An overview of the results is provided in Table 2.

[25] Cf. Microsoft (2021a).

Table 1: Classification Framework for ML-based AR Systems.

	Task	Hardware
AR	Which purpose does AR serve in the system (e.g. visualisation)?	On which (AR) device is the application deployed (e.g. mobile-iOS)?
ML	Which purpose does ML serve in the system (e.g. classification)?	Where is the ML model executed (e.g. server/cloud)?

The examined ML-based AR systems tackle a broad variety of problem domains like visual recognition, quality inspection and assembly. For example Eckert et al. (2018) and Shen et al. (2020) each created an application that aids the visually impaired by detecting objects in the vicinity of the users and – based on this information – provides them with audio guidance for navigation. The difference between the two applications is that Eckert et al. (2018) use cloud computing for their neural network execution, while Shen et al. (2020) let this step being performed directly on the device. Another example is Freeman (2020), who provides an ad-hoc translation and projection of a mobile weather forecast application from English to Chinese. Trestioreanu et al. (2020) and von Atzigen et al. (2021) tackle problems like image segmentation in radiology and intra-surgery object detection. However, the majority of authors that target a specific problem, focus their research on industrial applications. They improve quality control during[26] and after[27] assembly, help engineers make better design choices through generative algorithms[28], and support assembly processes by blending in text instructions[29] or superimposing 3D models on the product[30].

In addition to the papers and application domains already mentioned, there are other papers that deal exclusively with research on object detection for AR devices, such as Bahri et al. (2019) and Dasgupta et al. (2020). Both use a server to perform object detection and visualise the results on a HoloLens. Farasin et al. (2020) go a different way and implement an algorithm for tracking detected objects so as to reduce the need for subsequent neural network executions. Another interesting approach is the combination of ordinary object detection with spatial information in order to improve the detection results.[31]

[26] Cf. Krenzer et al. (2019).

[27] Cf. Wortmann (2020).

[28] Cf. Pullan et al. (2019).

[29] Cf. Svensson; Atles (2018).

[30] Cf. Su et al. (2019).

[31] Cf. Li et al. (2020).

Table 2: Literature on ML-based AR systems categorised systematically.

	Machine Learning		Augmented Reality	
Authors	Tasks	Hardware[32]	Tasks	Hardware
Bahri et al. (2019)	Object Detection	Cloud	Visualisation	HoloLens
Dasgupta et al. (2020)	Object Detection	Edge	Visualisation	HoloLens
Eckert et al. (2018)	Object Detection, Text2Speech	Cloud	Speech	HoloLens
Farasin et al. (2020)	Object Detection	Cloud	Visualisation, Tracking	HoloLens
Freeman (2020),	Translation	Cloud	Visualisation	Mobile
Krenzer et al. (2019)	Classification	Device	Feedback	Raspberry Pi
Li et al. (2020)	Object Detection	Device	Visualisation	Mobile (Android)
Liu et al. (2019)	Object Detection	Edge	Visualisation	Magic Leap One[33]
Pullan et al. (2019)	Classification, Anomaly Detection	Edge	Visualisation	Mobile (Browser)
Shen et al. (2020)	Object Detection, Text2Speech	Device	Speech	Mobile (iOS)
Su et al. (2019)	Pose Estimation	Edge	Projection	Mobile
Svensson and Atles (2018)	Object Detection, Segmentation	Device	Projection	Mobile (iOS)
Trestioreanu et al. (2020)	Segmentation	Cloud	Projection	HoloLens
von Atzigen et al. (2021)	Object Detection	Device	Projection	HoloLens
Wortmann (2020)	Object Detection	Device	Projection	Mobile (iOS)

Among 10 object detection tasks in the literature review, six choose HMDs as the AR device, while four target mobile devices. Out of these six detection tasks that target HMDs, only von Atzigen et al. (2021) execute the neural network on the device itself, a HoloLens 1. This is primarily due to the low computing capacity of HMDs, and thus a lot of research deals with the issue by off-loading the neural network execution to cloud or edge computers.[34] However, these solutions introduce other latencies like network traffic as well as making the system reliant on additional infrastructure, such as a stable network connection and the availability of servers. Therefore, having one device that successfully runs all necessary ML and AR operations should mostly be the preferred approach, especially during critical tasks, e.g. surgery.

[32] In case it was not clearly stated or ambiguous, on which hardware the ML model is executed, the most likely scenario was inferred.

[33] Liu et al. (2019) did not use the device istelf but a computer with the same specifications.

[34] Cf. Liu et al. (2019).

2.3 Object Detection with Deep Neural Networks

As became apparent in the section above, object detections plays an important role in AR, most likely because it is a prerequisite for real-world objects to be augmented with virtual information. We refer to object detection when the task at hand deals with detecting instances of visual objects of a certain class (such as humans, cats, or dogs).[35] Convolutional Neural Networks (CNNs), a neural network architecture that learns robust and high-level feature representations of images, are considered to be the most momentous approach for a variety of applications.[36]

Among the more important object detectors in recent history is the Faster Region-based Convolutional Neural Network (R-CNN), a two-stage detector that implemented a Region Proposal Network which allowed for nearly cost-free region proposal.[37] Two-stage detectors execute object detection as a divided approach, first proposing object regions and then classifying these regions.[38] With the model You Only Look Once (YOLO), Redmon et al. (2016) introduced the first single-stage detector architecture and achieved detection in real-time. They abandoned the region proposal and detection paradigm and instead applied a single neural network to the full image. This led to extreme improvements in speed albeit to some cost in accuracy. Figure 2 visualises the difference between one-stage and two-stage detectors on a high level. Not long after, the so-called Single Shot MultiBox Detector (SSD) combined the speed of YOLO and the detection accuracy of Faster R-CNN by predicting objects of different scales on different layers.[39] The detection accuracy of one-stage detectors was further increased with RetinaNet that introduced “focal loss” as a way to deal with extreme foreground-background class imbalance, which was the central cause of the inferior accuracy of one-stage detectors.[40] This approach was also adapted by Redmon and Farhadi in subsequent versions YOLOv2 and YOLOv3, which showed further improvements.[41] Most recent developments in CNN-based object detection that achieve good results at high speed are EfficientDet and the YOLOv4.[42]

[35] Cf. Zou et al. (2019).

[36] Cf. Sinha et al. (2017).

[37] Cf. Ren et al. (2015), Ren et al. (2017).

[38] Cf. Zaidi et al. (2021).

[39] Cf. Liu et al. (2016).

[40] Cf. Lin et al. (2017).

[41] Cf. Redmon; Farhadi (2017), Redmon; Farhadi (2018).

[42] Cf. Tan et al. (2020), Bochkovskiy et al. (2020).

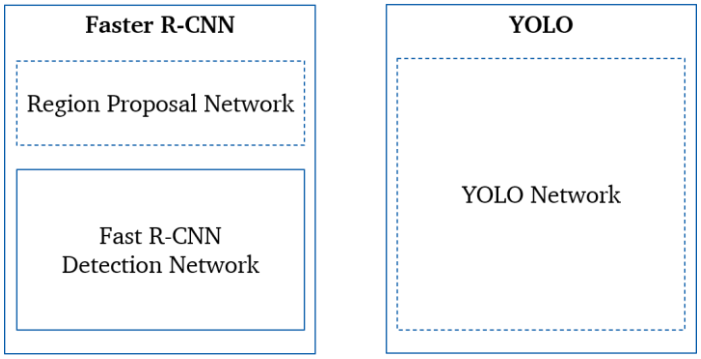


Figure 2: Two-stage detector with Region Proposal and one-stage detector with a single network.[43]

2.4 Research Gap and Research Question

Various approaches have been made to integrate neural networks into AR applications (Section 2.2). Since the computation capabilities of HMDs are still low, most of these solutions try to overcome the problem of high computational cost of neural network execution by off-loading it to external servers. While this is certainly a way to deal with the issue, it also has the downside of introducing more complexity into the system and making it dependent on additional infrastructure like network connections or third-party cloud servers. Furthermore, in some settings this may not be feasible, since factory buildings may lack a comprehensive wireless local area network (WLAN) or because the system needs to be fault tolerant under critical circumstances like surgery.

With the new frameworks available and rapid technological evolvement of both HMDs and object detection methods in mind, it seems a promising approach to aim for a ML-based AR solution that runs entirely on a HoloLens. (von Atzigen, et al. 2021) have done so by creating an application for the detection of screw heads in the context of surgical navigation. However, they evaluated their system against a custom dataset and can therefore not account for any loss of accuracy due to the technological framework of the HoloLens and the ML engine used.[44] This research gap can be closed by implementing a ML-based object detection system for the HoloLens that can be evaluated on established benchmark datasets. By doing so, the following RQs will be addressed:

1. What are the technical challenges of implementing a ML-based object detection system for the HoloLens?

[43] Adapted from Chen (2019).

[44] Cf. von Atzigen et al. (2021), pp. 6f.

2. How well do ML models perform in terms of accuracy and speed when used on a HoloLens compared to running them on conventional hardware?

3 Design Science Research Methodology

The research approach that was applied during the creation of the system follows the Design Science Paradigm, which puts forward the creation of a "viable artefact in the form of a construct, a model, a method, or an instantiation".[45] Gregor and Hevner (2013) define the term artefact as "a thing that has, or can be transformed into, a material existence as an artificially made object or process".[46] The contribution of such artefacts to knowledge can be expressed in terms of problem maturity and solution maturity. They can embody either known solutions applied to known problems (routine design), known solutions extended to new problems (exaptation), new solutions for known problems (improvement), or new solutions for new problems (invention).[47] While the solution maturity of detecting objects with deep neural networks is high, the maturity of the application domain is low. Therefore, positioning the research along these lines, the artefact presented here qualifies as exaptation research. Peffers et al. (2008) propose a Design Science Research Methodology (DSRM) Process Model that provides guidance for conducting and evaluating Design Science Research in information systems. The nominal process sequence in this model includes six steps, also referred to as activities. These are problem identification and motivation (1), definition of the objectives of a solution (2), design and development (3), demonstration (4), evaluation (5), and communication (6).[48] A summary of the DSRM Process Model applied in the context of the research presented here, can be found in Table 3.

While the DSRM gives guidance on "what to present", there remains the question of "how to present" it. For structuring and presenting DSR studies Gregor and Hevner (2013) provide a publication schema, which is itself based on work by DSR authorities.[49] The seven sections outlined in the schema are mirrored in this thesis and summarised here:

The thesis starts with an introduction (Section 1) that defines and motivates the problem in order to infer the objectives of a possible solution. At the beginning stands the observation that certain areas in industry lack the possibility for sufficient automation. Therefore, these areas rely heavily on human expertise, making their processes prone to error. This yields the need for solutions that support human workers, possibly with the problem-solving capabilities

[45] Hevner et al. (2004), p. 83.

[46] Gregor; Hevner (2013), p. 341.

[47] Cf. Gregor; Hevner (2013), pp. 344f.

[48] Cf. Peffers et al. (2008), pp. 11-14.

[49] Cf. Gregor; Hevner (2013), pp. 349-352.

of modern technologies like AR and ML. The subsequent review on literature (Section 2) does not only include research papers that are relevant to this study, but also highlight prior artefacts that have been developed to solve similar problems. In order to derive an artefact that combines AR and ML, it is important to define AR in distinction to VR, as both terms are often used ambiguously, and filter out prominent applications domains (Section 2.1). Furthermore, the question of what the capabilities of AR and ML are and how they can be combined needed to be addressed. A rigorous literature research for state of the art problems and solutions was conducted, resulting in a framework for categorizing such multi-modal systems (Section 2.2). This shows that object detection plays a vital role in ML-based AR applications, which is why recent state of the art object detectors are investigated (Section 2.3). Based on the resulting findings, the goal of this thesis could be formulated: the implementation of a system which executes a deep neural network for object detection on a HoloLens and makes use of the devices' visualisation capabilities (Section 2.4). Such a system can serve as a basis for solutions in other problem domains by exchanging the ML task and possibly also adapting the AR task. Whatever the combination of AR and ML, the system offers a maximum of flexibility for the user, as only the device is needed and no additional dependencies exist. In the method section (Section 3) the underlying research approach is outlined and explained with reference to existing authorities. The thesis follows the DSRM by Peffers et al. (2008) and is structured according to the publication schema by Gregor and Hevner (2013). Most noticeable and in line with the publication schema, the description of the artefact occupies the major part of the thesis and stretches over several sections. It starts with a description of the system and its architecture (Section 4.1) as well as the hardware (Section 4.2) and the development platform (Section 4.3). Next follows a thorough requirement analysis (Section 5) from a technical point of view, including a detailed description of the employed object detection model (Section 5.2). Section 6 describes the implementation of the systems and afterwards the artefact is evaluated by addressing appropriate criteria from both a quantitative and qualitative angle (Section 7). Peffers et al. (2008) also include demonstration of the artefact in this section, because the evaluation should show the solutions' ability to solve at least one instance of the problem. This demonstration happens in the form a benchmark test, which is the appropriate form of evaluation given that the artefacts' use lies in the future and will probably happen in the form of a different instantiation.[50] In addition to give evidence in terms of utility, semi-structured interviews were conducted considering the guidelines by Saunders et al. (2015). The

[50] Cf. Venable et al. (2016), p. 83.

benchmark and interview results are consequently discussed in terms of the implications for research and practice. In the discussion (Section 8) the main ideas of this thesis are recapitulated, highlighting possible further work as well as the limitations of the artefact before the thesis ends with a concluding paragraph (Section 9).

Table 3: Summary of the project in terms of the DSRM by Peffers et al. (2008).

Problem identification and motivation	Processes that cannot be automated and must rely on human expertise are prone to error. In these settings digital transformation is not about automation, but rather about assisting and increasing human performance.
Objectives of a solution	A system that combines human and machine capabilities through AR and ML can help to reduce the error rate. Performing object recognition on an HMD would be a promising base case for this and could be built upon for specific problem domains.
Design and development	A ML-based object detection system for the HoloLens is designed to assess the feasibility of neural network execution on HMDs. The general architecture of such a system includes access to sensor data, an inference engine for the ML model and the means to visualise the results. The instantiation uses the main camera of the device to access images of the user's surroundings, detects up to twenty different objects in them, and augments the real world environment with information about location and nature of these objects.
Demonstration and evaluation	The performance is validated by a test run on an object detection benchmark dataset. The utility is evaluated based on feedback from industry experts during an interview, where a live demonstration was given.
Communication	The main result of this research is the implementation of a neural network-based object detection system that runs on a HoloLens. It serves as a basis for the adaptation of other ML models in various problem domains. The results are communicated in this thesis and as part of a workshop with a technology company.

4 Artefact Description

The idea behind the ML-based AR system for object detection is to make use of the HMDs main camera and detect the objects in front of the user. The system can predict the objects' 2-D location as well as a score, that conveys how confident the algorithm is about the detection being correct. This information is then visualised in the field of view of the person wearing the HMD. The location is highlighted by a rectangle enclosing the detected object (a so-called bounding box). The name of the object and the confidence score are displayed at the bottom centre. Figure 3 shows some examples of the system in action.

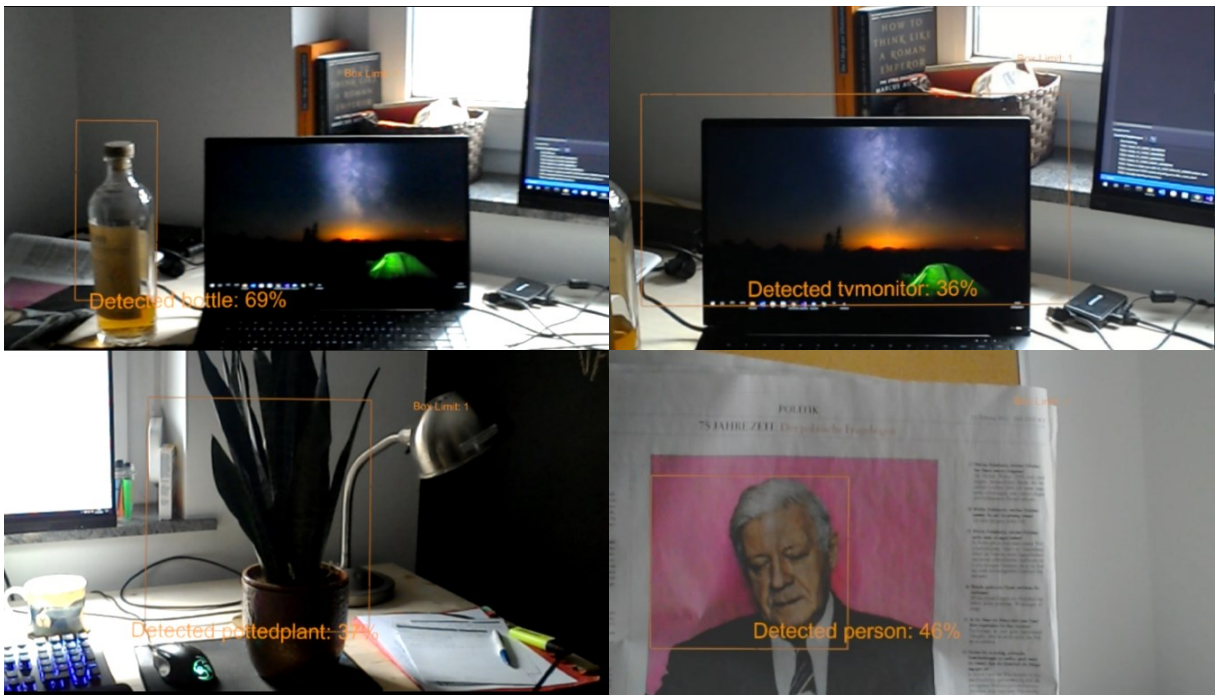


Figure 3: Demonstration of the object detection system. Detection of a bottle (top left), a monitor (top right), a potted plant (bottom left), and a person (bottom right).

4.1 System Architecture

In order to achieve this, the system can be divided into three components (cf. Figure 4): The first component is the Camera Module, which is responsible for accessing the camera's video feed. It can acquire individual images (so-called frames) from the feed that are needed for the object detection. The second component is the Detection Module. It takes a frame and subjects it to a pre-processing step. Subsequently, it executes an object detection model, which predicts the objects and their location in the frame. In a post-processing step this information is extracted and parsed into a standardised format for further processing. The third and final

component is the Visualisation Module. It takes the standardised information from the previous module, rescales them according to the size of the objects that have been detected and creates bounding boxes around the objects in the users' field of view.

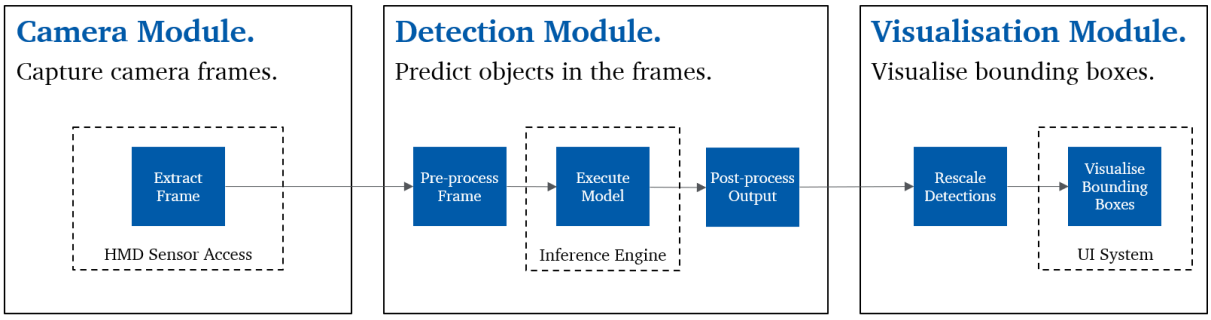


Figure 4: System Architecture Overview.

In contrast to similar systems that use a streaming solution and offload the object detection task to either the cloud[51] or an edge server[52], this system is executed on the HMD itself. This enables the system operator to move around freely and get immediate responses, even if there is no stable internet connection. The next section will discuss the choice of an HMD that provides the camera feed and runs the code of all three modules. This is followed by a description of the development platform that was used for the implementation of the system.

4.2 Microsoft HoloLens 2

While major companies like Apple are still working on their own mixed-reality devices, to date only few industry-ready HMDs exist.[53] Besides Microsoft's HoloLens, there is only the Magic Leap 1 whose hardware can keep up to some extent.[54] Because of its prevailing superiority, the prototype will be developed for the Microsoft HoloLens 2, which was released in 2019. It is the state-of-the-art mixed-reality device mostly focused on Industrial Augmented Reality applications, equipped with see-through holographic lenses. The camera can take pictures with up to 8-Megapixel (MP) and is also capable of Full HD video streaming at 30 frames per second (1080p30). HoloLens 2 can measure the distance to objects via a time-of-flight depth sensor and has awareness of body movement provided by an inertial

[51] Cf. Trestioreanu et al. (2020).
[52] Cf. Dasgupta et al. (2020).
[53] Cf. Mike Peterson (2021).
[54] Cf. Magic Leap (2021).

measurement unit (IMU). It gains additional understanding of its environment via six degrees of freedom positional tracking and spatial mapping. This information can be particularly helpful when objects not only need to be detected, but also tracked or localised.

Table 4: Overview of relevant HoloLens 2 technology specifications.[55]

Feature	Specification
Camera	8-MP still images and 1080p video streaming at 30 fps (16:9)
Depth	1-MP Time-of-Flight sensor
IMU	Accelerometer, gyroscope, magnetometer
Environment understanding	6 degrees of freedom tracking, real-time environment mesh
Processor	Second-generation holographic processing unit
Memory	4-GB LPDDR4x system DRAM
Battery life	2-3 hours of active use

HoloLens runs on the Windows 10 Holographic operating system, which uses the Universal Windows Platform (UWP) runtime environment. UWP helps developers to create applications that can potentially function on multiple types of devices like Windows, Xbox and HoloLens without having to change the code. It uses the Windows-Runtime (WinRT) API and supports programming in languages like C++, VB.NET, C#, F# and JavaScript.[56]

4.3 Unity Development Platform

A popular real-time development platform for creating virtual worlds is Unity.[57] It focusses on game development and comes with a lot of functionalities that can be helpful for AR applications as well. Furthermore, Unity supports building UWP applications for different architectures like x86 (HoloLens 1) and ARM64 (HoloLens 2). By default the platform generates a Visual Studio Project when building applications for UWP which has the possible advantage of including code written in e.g. C++ and thus include additional functionalities that might not be native to Unity.

Running a script in Unity executes a number of predetermined event functions, the so-called script lifecycle.[58] The lifecycle includes an initialisation phase, an update loop for animations (physics) and game logic, and the rendering of the scene. This is very important, because in

[55] Cf. Microsoft (2021a).

[56] Cf. Microsoft (2020a).

[57] Cf. Unity Technologies (2021a).

[58] Cf. Unity Technologies (2021b), Order of execution for event functions.

AR applications the scene needs to be updated constantly in order to react to changes in the environment. Considering, for example, the hologram of a ball rolling on the floor in front of the person wearing the HMD. Were there no updates to the scene, the ball would not move at all or – at a low update rate – move very strangely. This behaviour can also be observed in very old videos, where due to the low frame rate people appear to move very fast. However, the framerate in Unity can easily exceed 100 fps, which translates to one frame update per millisecond. In contrast, with the high computational demand of neural networks in mind, it is reasonable to expect the network to make about one prediction per second.[59] This presents a problem, since the detection model is part of the script lifecycle and would prohibit updates to the scene as long as the neural network is executing. Namely, a ML application that detects objects and has infrequent scene updates would show unintended behaviour (cf. Figure 5). Supposing, the HMD operator moves the detected object before the next scene update, then the bounding box would not adjust to the movement and be displayed at the same position in the field of view but not above the detected object anymore.

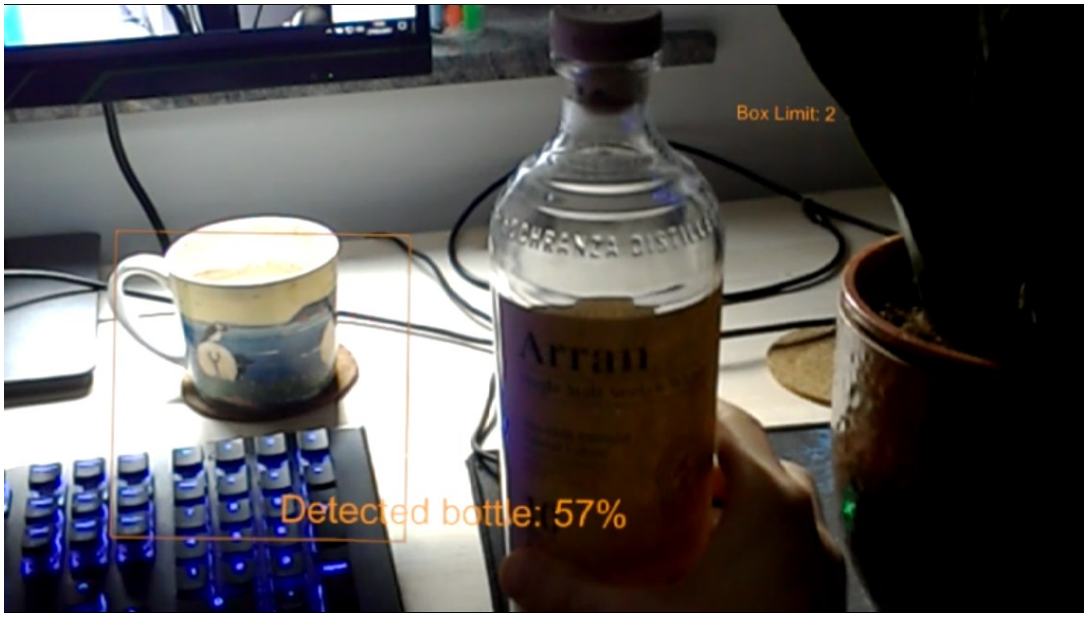


Figure 5: Unintended Behaviour: the detected bottle has been moved from its origin to the right between to frame updates, while the bounding box is still at the same position.

To mitigate this issue, the application will make use of Coroutines, a type of function that allows other parts of the code to run even though the computation inside the Coroutine has not finished yet. The principle is shown in Figure 6: A sequential execution (a) of the neural

[59] Cf. von Atzigen et al. (2021), p. 7.

network inside the update loop u_1 would cause a freeze of the application, seeing that Unity would not be able to perform the next update of the UI in u_2 before $prediction_1$ has been finished. Yet, enclosing the neural network in a Coroutine allows for an asynchronous kind of execution. This means that the script lifecycle is able to run recurring updates on the UI in u_1 and following. Only once $prediction_1$ is finished, the system resumes at the corresponding line in code and includes the result of the detection in the UI update u_5, putting the bounding boxes and labels in place.

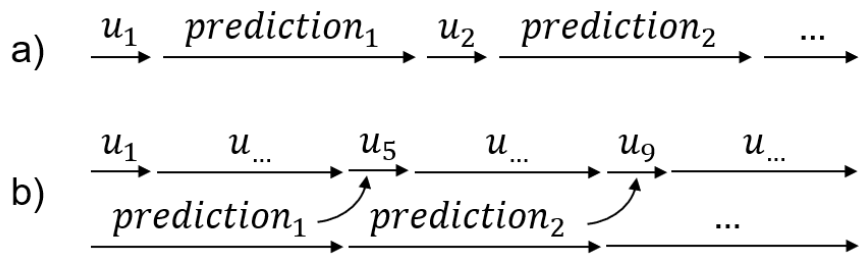


Figure 6: Sequential execution(a) vs. execution with Coroutines (b) in Unity.

The latest version of the ML-based object detection system for AR uses Unity version 2019.4.21f1 and is scripted in C#. Apart from the plugins mentioned in Section 5, no additional software was used.

5 Requirement Analysis

The modules have already been discussed on an abstract level. Before the system can be implemented, it is necessary to carry out a thorough analysis of the technical requirements. The specifics of the individual components must additionally be considered in detail, as there are dependencies between them that must be taken into account. More specifically, the next section will first outline how the Camera Module can access and extract frames (Section 5.1). Then follows a discussion of the available frameworks for the Detection Module to execute a neural network and make predictions on the HMD (Section 5.2.1). It is then possible to choose a detection model (Section 5.2.2), train it (Section 5.2.3) and analyse the necessary processing steps (Sections 5.2.4 and 5.2.5). Next is a discussion on how the resulting bounding boxes can be created by the Visualisation Module (Section 5.3). The chapter concludes with a section on how the system can be operated though basic gestures (Section 5.4) and a summary (Section 5.5).

5.1 Camera Module

There are multiple possibilities to access a live video feed from the HoloLens. These possibilities will be analysed and compared to choose the most suitable option for the object detection system. Table 5 gives an overview of the four different options of accessing the HoloLens. They are assessed on the basis of their capability to provide spatial information, on whether their parameters (e.g. framerate or resolution) can be adjusted, on the format that the images are delivered in, on whether they are native to Unity or provided externally, and lastly on their major drawbacks.

The straightforward option for accessing the camera is to use Unity's WebCamTexture class. WebCam Textures are textures onto which the live video input is rendered. The class has a devices property that allows to choose the camera that provides the live video. On the HoloLens this option defaults to the main RGB camera. Moreover, it is possible to set the frame rate (in fps) as well as the resolution with which the camera operates. What is not possible though, is locating the camera's position in and perspective on the scene. However, this is important in the real world for augmented imaging scenarios like the object detection system.

To map from camera coordinates to the real world coordinate system, Windows Mixed Reality provides a so called Locatable Camera.[60] This camera delivers spatial information for each frame and is easy to use in Unity via the VideoCapture class. However, it is not possible to access these frames because they cannot be streamed to memory. Instead VideoCapture only allows you to save videos or photos directly to disk.[61]

An open source project called CameraStream[62] addresses this issue by using the Windows MediaCapture API to feed the images' byte array along with the spatial information into Unity. The CameraStream plugin simply needs to be dropped in Unity and then mimics the VideoCapture class which makes its use straightforward. The downside is that on one hand, a third party or open source tool can make your application vulnerable. On the other hand, when using the VideoCapture class you will see a blinking camera icon in the field of view that can interfere with your projections. On top of that there can only be one VideoCapture instance alive at the same time, prohibiting the parallel use of the object detection application and other video related functions like recording a video or taking a snapshot.

Microsoft wants to help people use the HoloLens as a computer vision and robotics research device via HoloLensForCV.[63] With this repository they provide the means to access the devices' sensor streams that can be unlocked by enabling the research mode on the HoloLens. The research streams are coded in the programming language C++ and can be made available in Unity (which uses C#) through Intermediate Language To C++ (IL2CPP) Windows Runtime support.[64] This way it is possible to not only access the main camera but also the grayscale cameras, the depth camera and other sensors that might prove useful in further research. Enabling the research mode however has the major downside that it increases power consumption of the HoloLens, even when there are actually no running apps that use the sensor streams.

[60] Cf. Microsoft (2019).

[61] Cf. Unity Technologies (2021c), VideoCapture.

[62] Cf. Vulcan Technologies (2018).

[63] Cf. Microsoft (2020b).

[64] Cf. Unity Technologies (2021b), Windows Runtime support.

Table 5: Ways of accessing the HoloLens camera in Unity compared by their properties.

Camera Type	Spatial Info	Parameters	Format	Built-in	Drawbacks
WebCamTexture	No	Yes	ARGB32[65]	Yes	No spatial information
VideoCapture	Yes	Yes	N/A	Yes	Camera Icon
CameraStream	Yes	Yes	Byte	No	Camera Icon
HoloLensForCV	Yes	Yes	Byte	No	High Power Consumption

Taking the above considerations into account it is obvious that the VideoCapture class is not suitable for use in this project due to the lack of access to the individual camera frames. WebCamTexture seems like a good choice as it is already integrated in Unity and easy to use. However, the information about the cameras' position in the scene is missing. Even though this is not crucial for the artefact, having the corresponding knowledge about spatial conditions opens up a broader field for future work and research. Therefore, CameraStream and HoloLensForCV are the most suitable options. Both have similar features but the CameraStream plugin also has more drawbacks like the camera icon in the field of view. It also prevents the simultaneous use of the photo or camera function, which could still be useful. Therefore, despite the higher power consumption, the choice falls on HoloLensForCV provided by Microsoft. The next section discusses different ML backends that can be used to execute a neural network on the HoloLens.

5.2 Detection Module

5.2.1 Inference Engine

A challenge with the Microsoft HoloLens is that it has Windows 10 Holographic as the operating system, which is backed by Universal Windows Platform (UWP). UWP does not provide an integrated way of using ML models that have been created with standard frameworks like TensorFlow or PyTorch. Instead, they have to be converted to the Open Neural Network Exchange (ONNX) format and executed by additional libraries. Conversion, in the case of TensorFlow can be done with the appropriate tools.[66] PyTorch has an API for

[65] ARGB32: Alpha, Red, Green, Blue colour channels with 8 bit each.

[66] Cf. Open Neural Network Exchange (2021a).

exporting models to ONNX.[67] On top of that, the ONNX community provides a collection of pre-trained models for object detection in their Model Zoo.[68]

One way to integrate and execute these models in UWP applications on the device is to use the Windows Machine Learning (WinML) API.[69] WinML is an inference engine that allows to use trained ML models locally on Windows 10 devices. Like HoloLensForCV it is made available in Unity through Windows-Runtime support which in turn means that it is not possible to run it in the Unity Editor itself. Another way is to use the means provided by Unity directly. Unity-Technologies created their own lightweight inference library called Barracuda that is available via their package manager.[70] Barracuda works with ONNX models and fully supports CPU and GPU inference on UWP. It is not only easy to use but also being part of the Unity ecosystem makes it possible to run it in the editor, which can come in handy for testing purposes. Both options do not currently support all possible ML models. Even though WinML might turn out more flexible in terms of which model to use, the ease of integration and the possibility for in editor testing speak in favour of Barracuda.

5.2.2 Object Detection Model

Section 2.3 introduced two main types of neural network-based object detectors, one-stage and two-stage detectors. In general, one-stage detectors can reach higher detection speed than their counterpart, because they integrate the separate region proposal step into the detection network (cf. Figure 2). Because higher speed implies that for each detection less computations have to be executed, it is essential when dealing with low computational power. Thus, the system should rather incorporate a one-stage detector with only few layers than a deep two-stage-detector.

The ML inference engine Barracuda only supports a limited amount of neural network architectures, yet.[71] The one object detector that Barracuda can execute at the moment is the Tiny YOLOv2 model by Redmon et al. (2016). It is a light-weight network that achieves a lower precision than its “bigger brother” YOLOv2 (57.1% vs. 76.8% mAP), but it is more than

[67] Cf. PyTorch (2019), torch.onnx.

[68] Cf. Open Neural Network Exchange (2021b).

[69] Cf. Microsoft (2020c).

[70] Cf. Unity Technologies (2021d), Installing Barracuda.

[71] Cf. Unity Technologies (2021d), Supported Architectures.

three times faster (207 vs. 67 fps).[72] Thus, the model is adequate for the creation of a prototype that is supposed to run on a HoloLens, which provides only limited computation capabilities.

The network operates on input images of 416 x 416 pixels. By down sampling the images with a factor of 32 in the convolutional layers, the image gets divided in a 13 x 13 feature map. Feature maps are the output of the convolutional layers in the network and hold information about what is in the image (feature) and where it is in the image (map). Having a 13 x 13 feature map leads to an odd number of cells with only a single cell in the grids centre. The idea behind this is that especially large objects are usually found in the middle of an image. Having only a single cell instead of four cells representing the centre improves the prediction of such objects. For each of these grid cells, the model predicts five locations in the form of a rectangle, known as bounding boxes. They define the bounds of an object and are represented by an (x, y)-coordinate in the centre as well as the width and height of the box. Additionally to the spatial location, the model predicts a so-called objectness score and the class predictions for the objects itself. The objectness score expresses the probability that there is an object inside a predicted box. The class predictions are the conditional probability for each possible object, given that there is an object in the box. Given all this information, it is possible to infer the location and the nature of objects in the image. Figure 7 shows an image with the abovementioned cell grid and the final detection results represented by bounding boxes and labels.

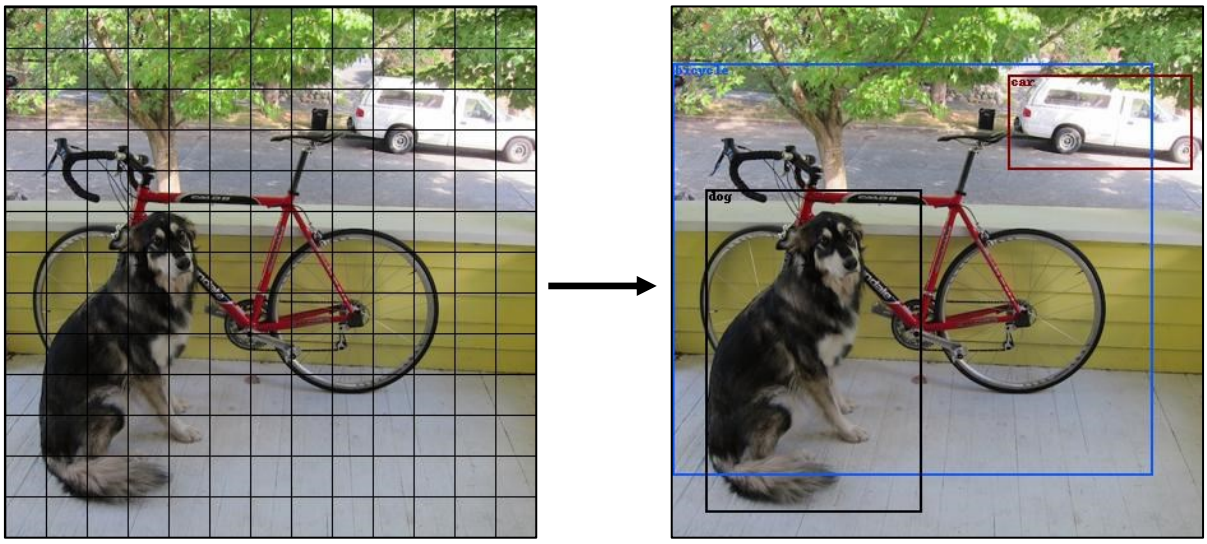


Figure 7: An image partitioned into 13 x 13 grid cells (on the left). The same image with bounding boxes and labels (on the right).[73]

[72] Adapted from Redmon (2016).

5.2.3 Training Dataset

Some models are better than others at detecting objects of a specific size.[74] However, the number and type of objects that can be detected are independent of the architecture of the model. Instead, it is during the training phase where this is determined. The Tiny YOLOv2 used in this thesis was trained on the PASCAL Visual Objects Challenge 2007 (VOC2007) and 2012 (VOC2012) training and validation datasets.[75] Like other popular datasets such as the Open Images Dataset[76] and the Common Objects in Context (COCO)[77], PASCAL VOC provides annotated (labelled) images that object detectors can be trained and evaluated on. Both datasets together have a total of 16,551 images containing 44,124 objects of 20 different classes. The full statistics are presented in Table 6. Being a relatively small dataset with a manageable number of object classes, it is well suited for the creation of a prototype.

Table 6: Object classes and statistics on the VOC2007 and VOC2012 training/validation data.[78]

	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car
2007							
Images	238	243	330	181	244	186	713
Objects	306	353	486	290	505	228	1,250
2012							
Images	670	552	765	508	706	421	1,161
Objects	954	790	1,221	999	1,482	637	2,364
	Cat	Chair	Cow	Dining table	Dog	Horse	Motorbike
2007							
Images	337	445	141	200	421	287	245
Objects	376	798	259	215	510	362	339
2012							
Images	1,080	1,119	303	1,286	1,286	482	526
Objects	1,227	2,906	702	1,541	1,541	750	751
	Person	Potted plant	Sheep	Sofa	Train	TV/Monitor	**Total**
2007							
Images	2,008	245	96	229	261	256	**5,011**
Objects	4,690	514	257	248	297	324	**12,608**
2012							
Images	4,087	527	325	507	544	575	**11,540**
Objects	10,129	1,099	994	786	656	826	**31,516**

[73] Adapted from Redmon (2016).

[74] Cf. Redmon; Farhadi (2018).

[75] Cf. Everingham et al. (2010), Everingham et al. (2015).

[76] Cf. Common Visual Data Foundation (2020).

[77] Cf. Lin et al (2014).

[78] Cf. Everingham et al. (2010), p. 310; Everingham et al. (2015), p. 107.

5.2.4 Pre-Processing

Pre-processing of the input for a convolutional neural network consists of two steps, resizing and normalisation. Resizing is necessary because models are trained and optimised for a certain image size and expect the input to be this particular size. If the images were of different size, the last convolution layer in the network would yield results that did not fit the expected dimensions for further processing.

Figure 8 shows three different ways of resizing an image to the corresponding size: the image can be resized as is, ignoring the original aspect ratio (1), or it can be cropped to a square before resizing, so the aspect ratio stays the same (2). Also the image can be resized while keeping the original aspect ratio but then needs padding on the shorter side (3). Ignoring the aspect ratio of the original image has the downside of distorting the real dimensions of the objects. In the first example of Figure 8 the resulting image appears to be squeezed and the objects do not have the same proportions as before. Even though CNNs are to a certain degree scale invariant, research has shown that scale and aspect ratio do matter especially when trying to detect small objects.[79] A method that keeps the aspect ratio is to crop a square from the original image. Having the same width and height, the images aspect ratio will not be affected by resizing and thus stay the same. A drawback of this method though is, that information will be removed from the image after cropping. This means that object, that could be seen before may not be included in the image after pre-processing anymore. For example, in the original image in Figure 8 it is possible to see a small red scooter in the top left corner, which is barely visible after cropping was applied. In the third method, the image is resized in a way such that the bigger side measures 416 pixels and the other side less. The aspect ratio stays the same and the smaller side gets padded with pixels to again have square dimensions. Compared to the other methods presented, this method does neither squeeze the image, nor does it remove information. However, the padded pixels yield no information whatsoever for the object detection task at hand but are processed anyway.

[79] Cf. Singh; Davis (2018), p. 3578; Qiu et al. (2019), pp. 1594f.

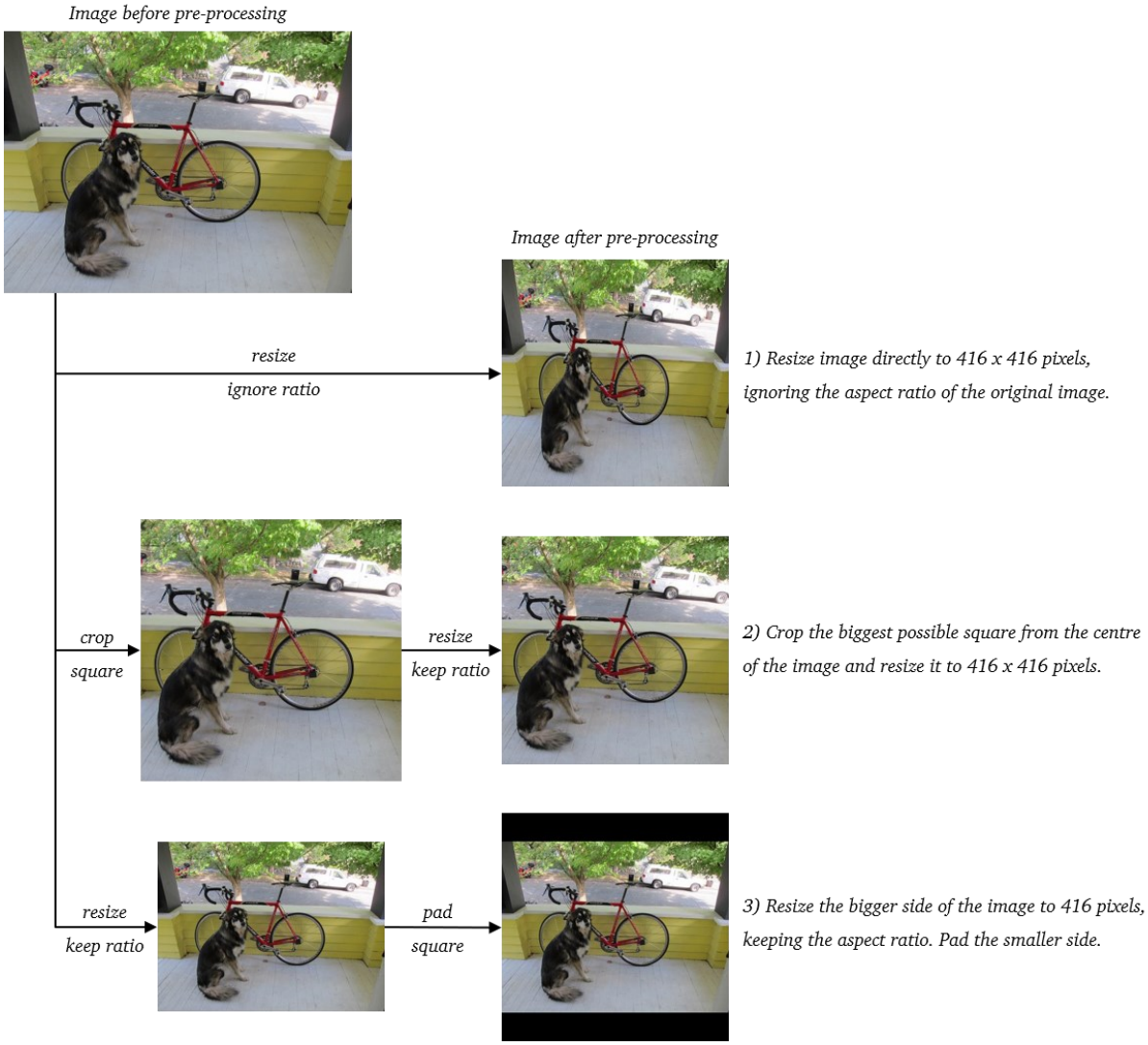


Figure 8: Illustration of different pre-processing methods applied to get an image shape of 416 x 416 pixels.

HoloLens provides images with a 16:9 aspect ratio. Resizing an image that is almost two times as wide as it is high with method one will result in big changes of scale and aspect ratio. At a first glance, the third method seems to be a more favourable alternative. With the restricted computational capabilities of the device in mind though, it is debateable whether a procedure that adds unnecessary data to the image is the right choice. On the contrary, it seems valid to assume that the loss of information with method one is rather minimal. Firstly, objects can often be found in the centre of the image.[80] Secondly, an object detection system on an HMD is unlikely to be used for objects on the edge of the field of view. Instead, users focus the

[80] Cf. Redmon; Farhadi (2017), p. 2.

object of interest and expect an analysis of this object and not some other objects at the sides. With the loss of information seemingly not very problematic, method two is a fast way of resizing the image and still keep the aspect ratio. Thus, it will be used in the implementation of the artefact.

The second step in pre-processing, normalisation, is an adjustment of the images' pixel values to the range between zero and one. It helps neural networks to converge much easier and thus makes the training process a lot faster and less complicated.[81] Typically, images are pre-processed with linear normalisation which transforms each pixel value v from the existing range (min, max) to a new range $(newMin, newMax)$. This method will also be used in the implementation and can be performed according to the following formula:

$$pixel = (v - min) * \frac{newMax - newMin}{max - min} + newMin$$

5.2.5 Post-Processing

As mentioned in Section 5.2.2, Tiny YOLOv2 partitions the image into a 13x13 grid and predicts five boxes for each of the grid cells. This makes a total of 845 bounding boxes per image. Hence, it is impractical to draw every bounding box, simply for reasons of space.

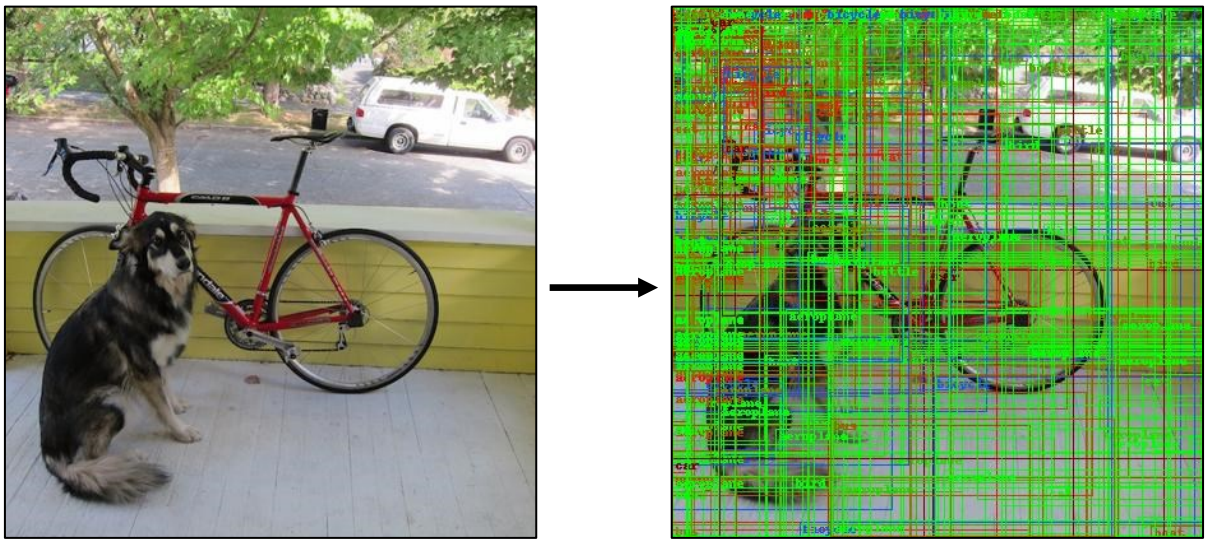


Figure 9: Result of the object detection. When no filtering is applied, all 825 predictions are visualised.[82]

[81] Cf. Liao; Carneiro (2016).

[82] Adapted from Redmon (2016).

Additionally, even though the model technically predicts an object for every box, it is more confident about some predictions and less confident about others. This leaves room for interpretation and when implementing such a model, it becomes necessary to decide at which level a prediction is accepted as sufficiently confident. At the same time, as can be seen in Figure 7 on the left side, objects usually stretch over multiple cells. Accordingly, the model predicts the same objects multiple times. Hence, it is necessary to find the bounding box that best represents each object. Figure 9 shows the result of the object detection algorithm without any filtering applied. In the rest of this section it will be explained, how false predictions can be filtered and how it is possible to decide which box represents an object best.

The filtering of bad predictions can be done by answering two questions. First, how confident is the model that this particular box contains an object? And second, assuming that there is indeed an object inside that box, how confident is the model that this object belongs to a particular class? To do so, it is necessary to look closer at the output of the model. For every box, the model predicts 25 features (cf. Figure 10), five of which describe the properties of the box (box features) and 20 of which are probabilities that decide on the object that is believed to be in the box (class probabilities).

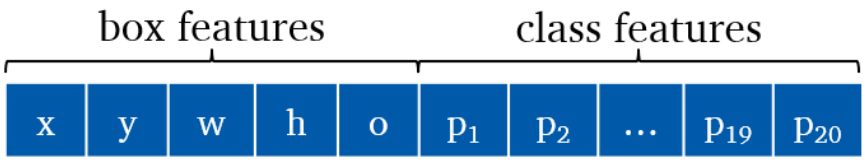


Figure 10: Tiny YOLOv2 output. For each bounding box, the models predicts 25 features: 5 box coordinates and a probability for each of the 20 objects in the dataset.

The feature *o* is called “objectness” and can be converted into a probability score by applying the sigmoid function to it. This probability score says, how likely it is that the box in question contains an object, thus answering the first question. Looking closer at the second question it becomes obvious that it also depends on the objectness score. “Assuming that there is indeed an object inside that box [..]” constitutes a probability under the condition that the box contains an object. This means that by multiplying the objectness score and the probability “that this object belongs to a certain class”, it is possible to answer question two. By applying a softmax function to the class features they can be converted into probability scores thereby telling how likely it is that the box contains an object of a certain class. The class with the

highest probability is multiplied by the objectness score, yielding the confidence score for that box:

$$confidence_{box} = objectness\ o * \max(class\ probabilities\ p)$$

Suppose the model predicts that a box has a 60% chance of containing an object ($o = 0.6$) and let the highest class probability of that box be $p_{12} = 0.75$. Applying above formula, the confidence score for that box is 0.45. In other words, the model is 45% certain that this particular box contains an object of class number 12 (a dog in the Pascal VOC dataset).

By setting a threshold value above which the detection is recognised as sufficiently good, boxes that fall below the threshold value can be sorted out. By default, Tiny YOLOv2 only displays objects detected with a confidence of 25% or higher. The threshold for objectness and confidence can be adjusted freely though. Setting a higher value yields better quality but less quantity of detections. Quality meaning that the detected objects are more likely to be correct predictions, while quantity refers to the number of detected objects both correct and false predictions. A threshold of zero would display all detections while a threshold of one would very likely display none at all.

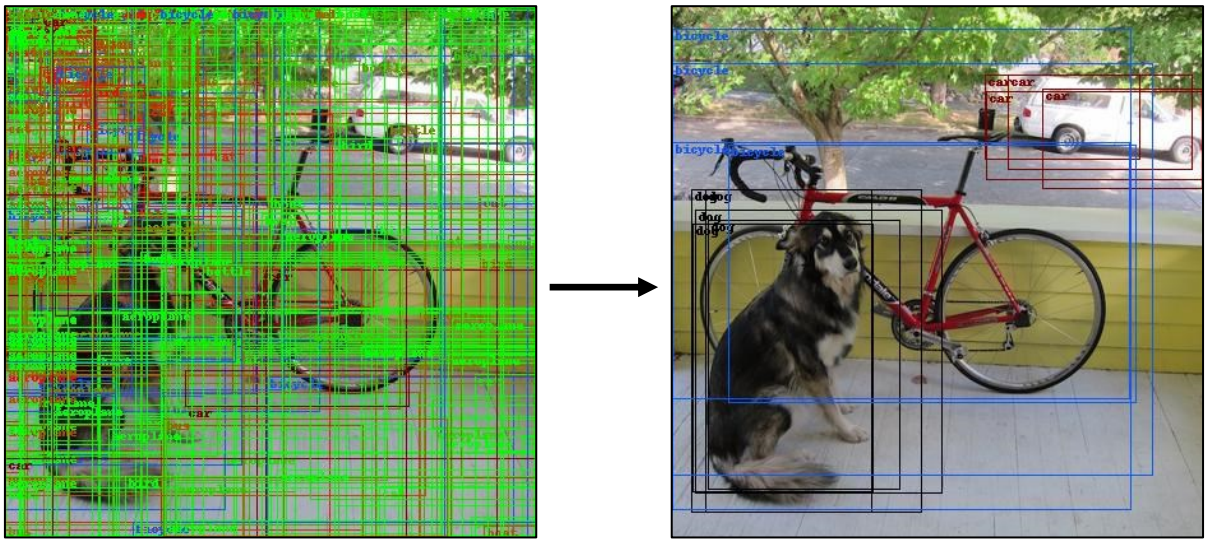


Figure 11: Result of the object detection after filtering with a confidence threshold of 25%.[83]

The need for a second filtering step results from the two facts that Tiny YOLOv2 predicts five boxes per grid cell and that an object can stretch over multiple grid cells in the image. Two or

[83] Adapted from Redmon (2016).

more neighbouring cells can thus predict boxes that are eventually describing one and the same object. A common method that helps dealing with this issue in object detection is called non-max suppression (NMS). NMS compares bounding boxes by measuring the similarity between two boxes. The measure of similarity for two overlapping regions is the Intersection over Union (IoU) ratio, also referred to as the Jaccard Index.[84] It ranges from zero to one and the higher it is the more similar the boxes are. IoU can be calculated by measuring the overlap between two boxes over their union:

$$IoU = \frac{area_{box1} \cap area_{box2}}{area_{box1} \cup area_{box2}}$$

If the IoU exceeds a certain threshold, the boxes are considered to describe the same object and only the one with the higher confidence is kept while the other is discarded. Per default Tiny YOLOv2 works with a threshold of 0.45 IoU.

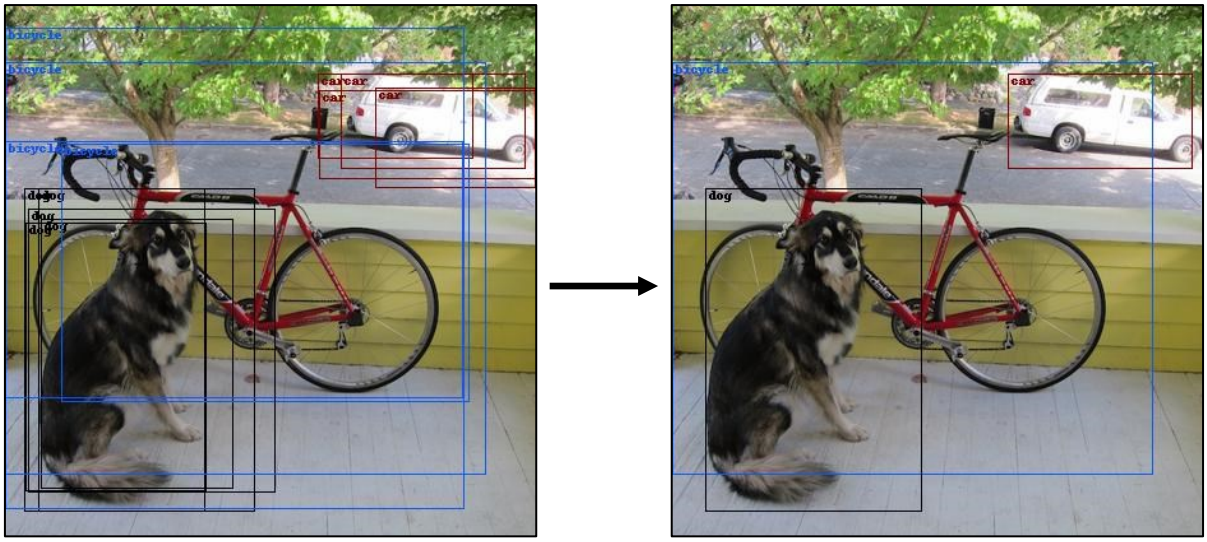


Figure 12: Result of the object detection after NMS has been applied with an IoU threshold of 45%.[85]

In summary, the post-processing applies two filtering steps: in the first step, bounding boxes are discarded based on the objectness score and the class probabilities. In the second step, boxes can be filtered based on their similarity, which is indicated by the IoU ratio. The remaining boxes represent the objects that have effectively been detected.

[84] Cf. (Jaccard 1901).

[85] Adapted from Redmon (2016).

5.3 Visualisation Module

When visualising bounding boxes, the boxes should appear between the user and the detected object, enclosing the object with a rectangle (see Figure 3 for an example). Unity has three UI systems that could be used to draw bounding boxes: UI Toolkit, Unity UI (uGUI), and Immediate Mode GUI (IMGUI). In the documentation they provide an exhaustive comparison of these systems.[86] A comparison by type of UI, whether developing a UI for the Editor or a runtime UI for applications is shown in Table 7.

Table 7: Unity UI systems compared by type of UI.

Type of UI	UI Toolkit	Unity UI (uGUI)	IMGUI
Runtime	Yes (preview)	Yes	Not recommended
Unity Editor	Yes	No	Yes

The IMGUI system serves as a code-driven interface. Instead of placing permanent game objects in the scene, the code in its OnGUI function is executed and drawn into the scene at every frame. This behaviour closely mimics what 2D object detectors do and would be well suited to drawing bounding boxes on the surface. Apart from not being recommended for use at runtime, the IMGUI system does not work in mixed reality applications. That leaves two options: the basic uGUI or the future go to UI Toolkit, which is still in a development stage. The latter is optimised to deliver high performance. That makes it a good choice for an application that already needs a lot of computational power to execute a neural network. However, it does not yet support World-Space rendering which is needed for the creation of the bounding boxes. Therefore, using the uGUI system is the only choice.

Now that the necessary components have been chosen, it is time to have a look at how the application can be operated. Having basic control over the functions is not only useful for debugging and testing, but also necessary with regard to potential future industrial applications.

5.4 Operating the System

Being able to operate the object detection system mainly serves two practical purposes. First, it allows the user to regulate the number of objects to be visualised. By adjusting the so called box limit, one can determine the maximum number of boxes that appear in the field of view

[86] Cf. Unity Technologies (2021e), Comparison of UI systems in Unity.

after each detection phase. While certain use cases may only need one box others may need to detect as many objects as possible. Secondly, being able to start and stop the object detection without closing the whole application, makes it easier to perform intermediate tasks without having the holograms present.

Table 8: Gestures and corresponding actions for control of the application.

Gesture	Action	Applicability
Single Air tap	Increase Box Limit	While detection is running
Double Air tap	Start/Stop Detection	When stopped or running

HoloLens applications can be controlled by either voice or gestures. Voice commands can save time and do not need any additional hand movement. Implementing custom voice commands however comes with some overhead. With future industrial use in mind, one can also argue that in a loud environment like a factory they might not be very reliable anyway. Hand gestures on the other hand can be very reliable as long as they are not demanded for other tasks like holding tools. But it seems fairly safe to assume that changing the box limit and starting or stopping the application are actions that are rather performed before or after a task has been completed.

There is only one basic hand gesture that comes into question, the so-called air tap (cf. Figure 13). To perform an air tap, the user has to hold his hand straight out in front of him in a loose fist and point his index finger straight up, which initialises the command. Tapping the finger down once and then quickly raising it back up again executes the air tap. By tapping again, the user performs a double tap gesture. By assigning actions to these gestures the application can be operated. A single tap increases the box limit by one and up to a maximum of five. Then the counter starts again from the beginning. A double tap will start the system or stop it when it is already running.

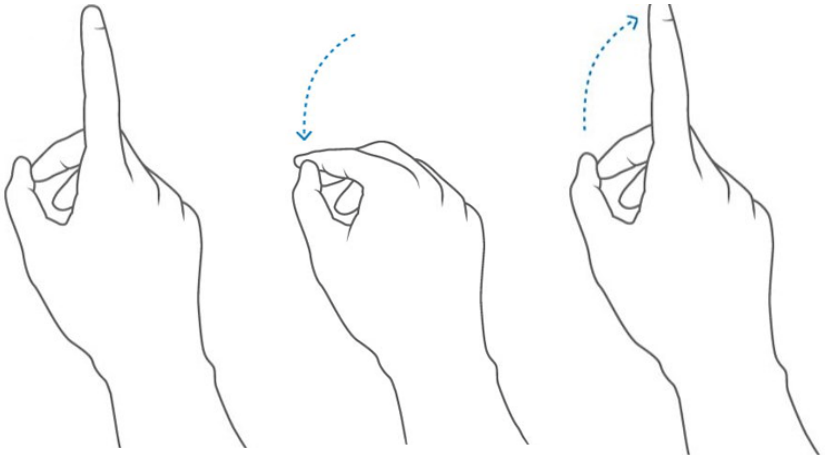


Figure 13: HoloLens Air Tap Gesture: hold finger in ready position, press down and lift back up.[87]

5.5 Summary

In this section the technical requirements have been analysed and different solutions for frame extraction, model execution and bounding box visualisation have been evaluated. The Camera Module can access frames via HoloLensForCV and the Detection Module makes use of Unity's Barracuda inference engine for execution of the detection. The Unity UI package provides the means for the creation of bounding boxes by the Visualisation Module. An overview of the system architecture with these components is presented in Figure 14. Meanwhile, the system can be operated by basic air tap gestures, changing the box limit, and starting or stopping the detection system.

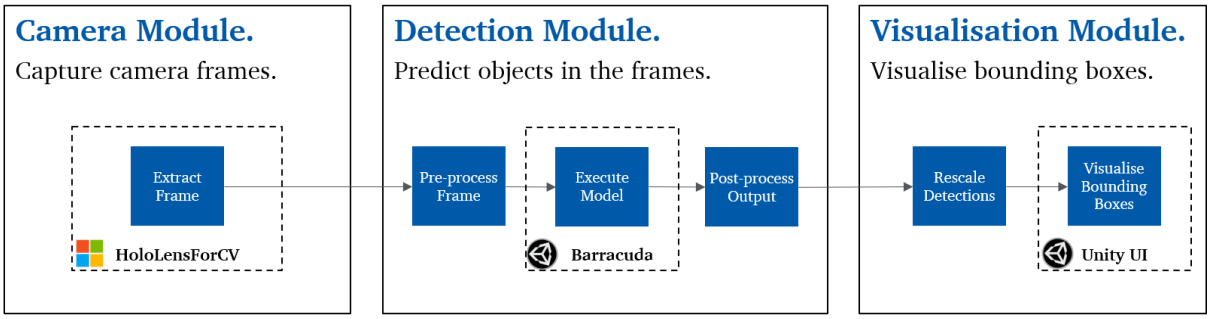


Figure 14: System Architecture Overview with components.

The model detection model that will be used in the implementation is the Tiny YOLOv2. It has been trained on the VOC2012 data and can detect twenty different object classes. It works on

[87] Cf. Microsoft (2020d).

images that are 416 pixels wide and high and have values in the range between 0 and 1. The result of the model is a 13 x 13 grid with five bounding boxes in each cell. The bounding box coordinates are predicted in relation to their cell and need to be transformed into relative values. Bounding boxes are then filtered twice, first by confidence and second by IoU. The final boxes are the ones to be visualised.

6 Artefact Implementation

The requirements being defined, and the components of the application being chosen, everything is set for the modules (cf. Figure 4) to be implemented. The following sections will discuss the particularities of the corresponding code and explain how the implementation works. The first part of the architecture is the camera module. Access to the camera needs to be established (Section 6.1) and the pre-processing steps need to be implemented (Section 6.2.1). After that, the pre-processed images serve as the input for the execution of the detection model (Section 6.2.2). The output of the model needs several steps of post-processing (Section 6.2.3). Then follows a description of how the final results are visualised (Section 6.3) and an explanation of how the system can be operated (Section 6.4), all followed by a brief summary (Section 6.5).

6.1 Camera Module

The first part of the implementation is to obtain a live camera feed from the main front-facing Photo/Video (PV) camera. As discussed in Section 5.1, the camera stream can be accessed using HoloLensForCV. This plugin has to be integrated in Unity via Windows Runtime support because it contains code that does not work in the Unity Editor. In order to do so, all the code that uses HoloLensForCV has to be enclosed by a directive that checks if Windows Runtime support is enabled (cf. Figure 15). The idea of this directive is that before calling a Windows API from a script, it has to be ensured that the script can actually be run by the environment that tries to execute it. If the latter is not the case, like in the Unity Editor, then the directive makes the system ignore this part of the script.[88] Otherwise the development environment detects compile errors, which prevents building the application. Therefore, it would not be able to create a visual studio solution for the installation on HoloLens.

```
using System;
using UnityEngine;

#if ENABLE_WINMD_SUPPORT
using HoloLensForCV;
#endif
```

Figure 15: Import of standard namespaces and HoloLensForCV enclosed by a #define directive.

[88] Cf. Unity Technologies (2021b), Windows Runtime support.

Before the system starts streaming the camera frames, the video stream has to be initialised with the necessary configurations. The access point for streaming sensor data is the MediaFrameSourceGroup object. It gets created with its properties set to stream frames from the PV camera and can subsequently be enabled and then started. The great thing about this object is that it cannot only stream PV data but also any other sensor information e.g. depth data. Additionally, it provides the possibility to set camera properties like the camera resolution and the rate at which frames stream in. For now the system uses only the PV camera with the standard properties. This means frames arrive at a rate of 15 to 30 fps with a resolution of 869 pixels in width and 504 pixels in height.[89]

Once the MediaFrameSourceGroup has been started, individual sensor frames can be accessed. In order to do so, the system uses Unity's OnUpdate function. The MediaFrameSourceGroup gets prompted for the latest frame and once arrived, the frame will be converted to be in a Unity viewable bitmap format. This bitmap in turn can be converted to a byte array which is necessary to load the data into a Unity Texture object. This is the final data structure before the frame is going to be pre-processed.

Recalling the script life cycle presented in Section 4.3, the OnUpdate function gets called once per frame update. The execution of the object detection model stretches over several frame updates though. As mentioned above the camera operates at up to 30 fps but the detection system manages only around one fps. By scheduling 30 frames for processing, while at the same time only one frame can be processed, the system would perform a lot of unnecessary computations and display unwanted behaviour. To overcome this issue, a new frame will only be acquired once the current frame has been fully processed. The check, whether the system is still processing a frame, can be made in the OnUpdate function as well. Unity's update routine runs even faster than 30 fps, so after having processed a frame, the check will happen very soon without spending much time idling. An overview of the image retrieval processes is presented in Figure 16.

[89] Cf. Microsoft (2019).

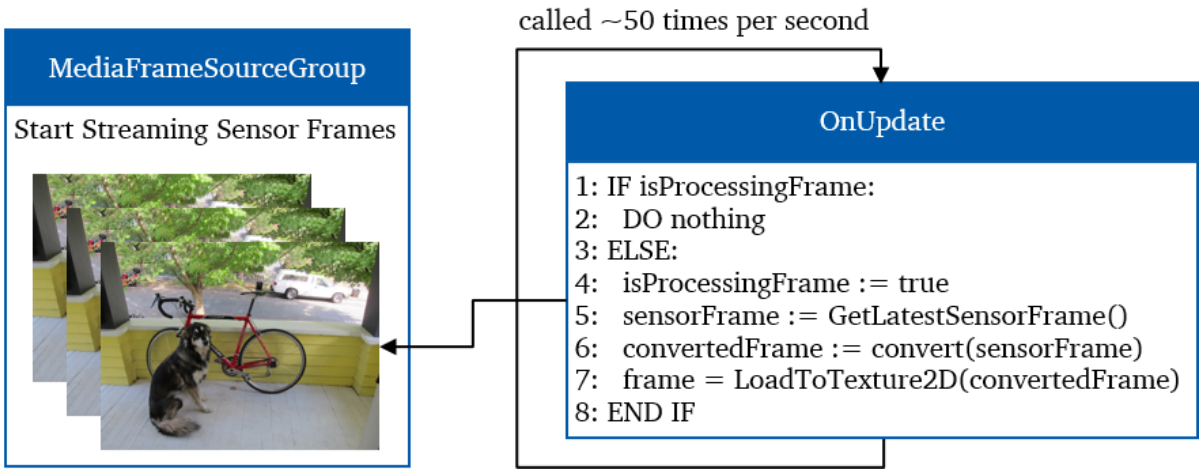


Figure 16: Image Retrieval Implementation Overview with Pseudo Code.

It is noteworthy that transforming the frame into a Texture2D before the pre-processing is probably not the most efficient way of handling incoming frames. The integration of additional plugins like the image processing library OpenCV could possibly be perform pre-processing on the bitmap format and speed things up. However, with Texture2D being a standard Unity format, it has the advantage that the code is more modular. The pre-processing step will expect a Texture2D as input and always work with this format even when the sensor stream is accessed in a different way. This is possible since all frame acquisition methods presented in Section 5.1 can fall back on the Texture2D format. In contrast not every way of accessing the camera stream can for example provide images in byte format.

6.2 Detection Module

6.2.1 Pre-Processing

Having loaded the frame in a Texture2D the system can start with the two first steps of pre-processing, cropping and resizing (cf. Figure 17). However, these operations are not very straightforward in Unity. A built-in cropping function does not exist and the Resize function from Texture2D just resizes the texture itself but not its contents. Instead, it even sets the pixels to be undefined, making the image grey. Yet, it was not necessary to implement these

functions from scratch. Instead, third-party code for texture handling could be adapted for the requirements of this detection system.[90]

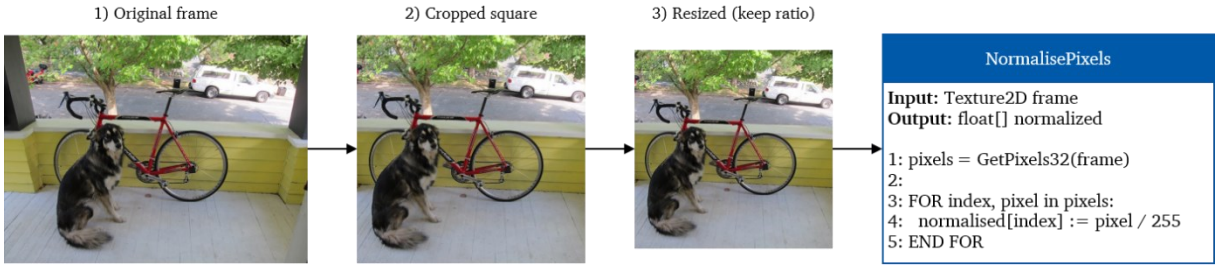


Figure 17: Pre-Processing Implementation Overview with Pseudo Code.

Before it is possible to apply normalisation to the frame, the pixel values need to be extracted. By calling the GetPixels32 function on the texture, Unity returns these values in Color32 format, a representation of RGBA colours in 32 bit. Hence, each colour component takes on a value in the range 0 to 255.[91] Recalling the formula for normalisation from section 5.2.4, this is an important fact because it is possible to infer that min = 0 and $max = 255$. In the same section it was already discussed that the target range lies between 0 and 1, hence $newMin = 0$ and $newMax = 1$. Inserting these values into the formula, yields the following transformation:

$$pixel = (v - 0) * \frac{1 - 0}{255 - 0} + 0 = v * \frac{1}{255}$$

So in order to normalise the pixels, the values from the Red, Green, and Blue channels simply have to be divided by 255, while the Alpha channel can be discarded. With all pre-processing steps performed, the frame is now ready for the object detection step that is described in the next section.

6.2.2 Model Execution

In Barracuda the model does not accept raw pixel values as input, so the pixels have to be embedded in a tensor of the shape *(1, 416, 416, 3)*. The first dimension describes the number of input frames (batch size), the second and third represent the frames' height and width respectively, and the fourth dimension is the number of channels. This is also referred to as

[90] Cf. Ashikhmin (2020), TextureTools Script.

[91] Cf. Unity Technologies (2021c), Color32.

the NHWC format in distinction to number, channel, height and width (NCHW). With the model expecting a three channel input, it becomes clear why the alpha channel has been discarded during pre-processing.

Before the model can be executed, the inference engine has to be initialised in the form of the Barracuda IWorker interface. This worker breaks down the model into executable tasks and schedules them on GPU or CPU, depending on the specified backend worker type.[92]

```
IWorker worker = WorkerFactory.CreateWorker(workerType, model);
```

It has already been noted that running the model is a computationally expensive operation and that the Unity Update routine will not continue until all sequential operations have been completed (cf. Section 4.3). One way to overcome this limitation is to use coroutines. A coroutine performs calculations without blocking other functions as long as they do not depend on their results. By enclosing the model execution in a coroutine, Unity can perform any UI updates necessary while waiting for the result. The use of a coroutine is indicated by the IEnumerator keyword and it allows other processes to run as soon as the yield-return-statement is reached. So after the tensor with the input frame has been created the detection coroutine starts the inference process and immediately returns to next line of code that can be executed independently. Only one the inference step is finished, executions return to the coroutine and fetches the detection results from the worker. The pseudo code of the coroutine is presented in Figure 18.

DetectionCoroutine

Input: float[416 * 416 * 3] pixels
Output: Tensor result

1: tensor = CreateTensorFrom(pixels)
2: START yield return worker.execute(tensor)
3: result = FetchOutputFrom(worker)

Figure 18: Execution of the Detection Model Pseudo Code.

Once the neural network has been successfully executed, the program returns to yield return-statement and fetches the output of the inference step, ready for post-processing.

[92] Cf. Unity Technologies (2021d), Using IWorker Interface.

6.2.3 Post-Processing

Depending on the architecture of the model that was used for making predictions, the output must be post-processed differently. Tiny YOLOv2 divides images into a 13x13 grid and predicts five boxes per grid with 25 features for each box. This results in an output Tensor of the shape *(1, 13, 13, 5*25)*. The first step in post-processing (cf. Figure 19, on the left) is to iterate over every cell in the grid and filter the boxes with low confidence values. The threshold for this confidence value is set to 25% (or 0.25) in the implementation. For boxes that exceed the confidence threshold, the coordinates and the class probability are extracted. Subsequently, the coordinates need to be scaled, such that the centre coordinates *x* and *y* as well as *width* and *height* of the box are relative values between 0 and 1. That way they can later be converted to absolute values corresponding to the size of the original frame by multiplying them with the original frames' width and height. These boxes are then added to a list, which is the output of the first post-processing step.

The second step in post-processing is NMS (cf. Figure 19, on the right), which filters the remaining boxes based on their overlap. To do this, the list of boxes needs to be sorted by confidence in descending order, the first box in the list being that with the highest confidence value. Now the area of the first box is compared to the area of all other boxes by calculating the IoU. If the IoU value is higher than a certain threshold (set to 0.45 in the implementation) the box with the lower confidence value is discarded, assuming that both boxes describe the same object. This process is then repeated for all the boxes that have not been discarded until only boxes remain that have the highest confidence value for a certain object.

Step1: ParseOutput

Input: Tensor*(1, 13, 13, 125)* result
Output: List[BoundingBoxes] boxes

```
FOR cell in grid
  FOR box in cell
    IF box confidence < threshold
      GOTO next box
    ELSE
      SCALE box coordinates
      ADD box to boxes
    END IF
  END FOR
END FOR
```

Step 2: NonMaxSuppression

Input: List[BoundingBoxes] boxes
Output: List[BoundingBoxes] filteredBoxes

```
sortedBoxes = boxes.sortBy(confidence)
FOR boxA in sortedBoxes
  ADD boxA to filteredBoxes
  FOR every other boxB in sortedBoxes
    IF IoU(boxA, boxB) > threshold
      DISCARD boxB
    ELSE
      GOTO next boxB
    END IF
  END FOR
END FOR
```

Figure 19: Post-Processing Implementation with Pseudo Code.

Since the number of boxes may still be relatively high, depending on the use case it can make sense to reduce it further. This can be done by introducing a limit for the maximum number of boxes. Suppose a quality inspection scenario where the product is scanned for cracks. The model is then trained to detect different types of cracks, with one crack being sufficient for the product to be sent back for refurbishment. Setting the box limit to 1 allows the system to detect the crack with the highest confidence and stop the NMS at this point, avoiding further computations. However, this is not a part of the original Tiny YOLOv2 implementation because it would be contradictory to the nature of general object detection task, trying to detect all the objects in an image. In the implementation the box limit can be adjusted by the user while the system is active to allow for maximum flexibility.

6.3 Visualisation Module

As briefly described in Section 4, his module is planned to mimic the behaviour of typical 2D object detectors, which place a bounding box on top of an image or video frame. Following the uGUI guidelines for creating a World Space User Interface, a canvas of 800x450 pixels size is placed in front of the camera.[93] A text with information about the detected objects' name and confidence score is located at the bottom centre of the canvas. The current box limit is positioned at the top right, and the bounding boxes will be rendered wherever an object was detected (cf. Figure 20). Because rendering content too close to the user can be uncomfortable in mixed reality, the UI is one metre away from the camera.[94] This is a trade-off between the 0.85 metres recommended for the nearest rendering and the 1.25 metres, where the optimal zone for ideal hologram placement begins, according to the Microsoft rendering guidelines.[95]

[93] Cf. Unity Technologies (2020), Creating a World Space UI.

[94] Cf. Microsoft (2021b).

[95] Cf. Microsoft (2020e).

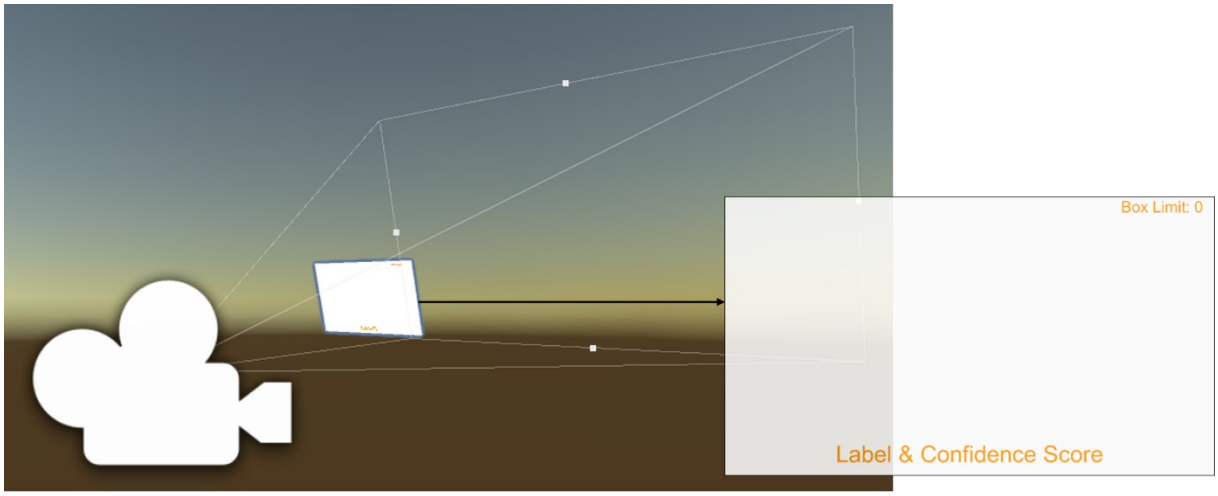


Figure 20: Unity Editor View. The user's perspective on the canvas (left) and an enlarged view of the canvas (right).

The bounding boxes arrive at the visualisation module with normalised, relative coordinates. They are adjusted to the size of the canvas by multiplication with its height and width. Furthermore, x and y need to be changed from representing the centre of the box to representing the bottom left corner. This is necessary to work with the drawing function. Before drawing, the system performs two final checks. The first check makes sure that the boxes are at least 25 pixels wide and high, to avoid drawing particular small predictions. The second one is a boundary check, verifying that the box dimensions do not exceed the size of the canvas they are rendered on. If so, the size of the box is decreased to match a certain offset from the boundaries of the canvas. The offset is here is set to 2 and helps avoiding that the bounding box wraps around the edges. An overview of the visualisation can be found in Figure 21

```
DrawBoundingBoxes
Input: List[BoundingBoxes] filteredBoxes

1:  FOR box in filteredBoxes
2:    SCALE box to texture.size
3:    IF box.size < 25
4:      GOTO next box
5:    ELSE IF box wider than canvas
6:      box.x_max := canvas.width – 2
7:    ELSE IF box higher than canvas
8:      box.y_max := canvas.height – 2
9:    END IF
10:   DRAW box
11: END FOR
```

Figure 21: Bounding Box Drawer Pseudo Code.

Up to this point, only two dimensions, namely the width and height of the objects in the images, have been considered. This will lead to inaccurate bounding boxes in a 3 dimensional scenario though, where the object is further away from the user than the canvas holding the bounding boxes. This difference in perspective has the effect that the bounding boxes would be too small and not fitting the object if they were not scaled before drawing them on the canvas. In order to do this properly a lot of transformations have to be made. For the purposes of this implementation the scaling parameters have been obtained in an experiment, where the size of the canvas has been altered until the bounding boxes approximately fitted objects in one to three meters distance.

6.4 Operating the System

In section 5.4 two gestures were proposed to operate the system. A single air tap to change the box limit and a double air tap to start and stop the detection (cf. Table 8). To recognize these commands a Gesture Recognizer object has to be set up for Tap and DoubleTap and be initialised before the detection can start. While the application waits for the DoubleTap gesture, the box limit cannot be changed. Only once the user performs a DoubleTap, the camera stream gets loaded, detection starts with the first frame that is acquired, and the box limit is set to one. Now, when a Tap Event occurs, the box limit will be increased in increments of one. Once it reaches five and another Tap Event is recognised, the box limit

counter starts again at one. At any point in time can the detection be stopped by another DoubleTap gesture.

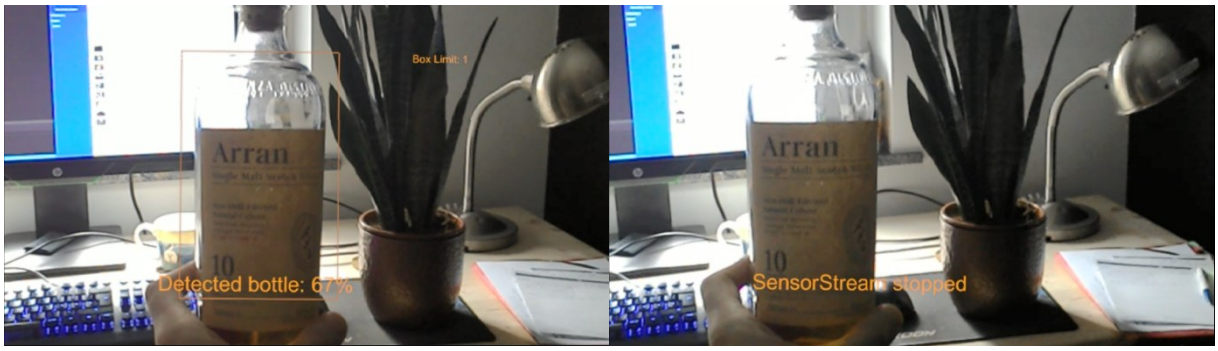


Figure 22: Detection with a box limit set to one (left) and the system after the the detection has been stopped by a double air tap gesture (right).

6.5 Summary

The camera operates at 30fps providing images at a resolution of 896x504 pixels. Before being fed forward into the network, each image is resized to match the input requirements of the Tiny YOLOv2 network which-is 416x416 pixels. Subsequently Barracuda executes the neural network, which divides the image into 169 grids of equal size and predicts five bounding boxes per grid. Each bounding box has a probability assigned to it which tells how likely it is that the box contains an object. When this probability score exceeds a certain threshold the object and the bounding box coordinates are extracted during the post-processing phase. The boxes can then be compared and those with a big overlap are excluded to avoid multiple detections of the same object. Afterwards, the visualisation module takes the filtered boxes, scales them according to the size of canvas and draws them on a transparent 2D plane in the users' field of view.

7 Evaluation

The artefact is evaluated in terms of validity and utility. Validity means that the artefact works and does what it is meant to do.[96] It is dependable in operational terms in achieving its goals. Utility, on the other hand, is more of a qualitative nature. It assesses "whether the achievement of goals has value outside the development environment".[97] The validity of the system can be evaluated by performing a quantitative analysis of the object detection results. This involves a description of the required metrics and analysis techniques (Section 7.1).[98] To show the utility of the project, the possible applications of the system outside of the development environment will be substantiated by interviews with industry experts (Section 7.2). The goal of the interviews is to discuss the justification for the proposed combination of AR and ML and subsequently talk about further applications of the system in various domains.

7.1 Object Detection Benchmark

7.1.1 Test Dataset

The first part of the object detector evaluation will be performed on the VOC2007 test data. The PASCAL VOC challenge was for some time considered *the standard* benchmark in object detection and is still very popular, among other datasets like the more recent Microsoft COCO.[99] Evaluating the project on this particular dataset is straightforward because it has also been used to train the model (Section 5.2.3). More important it has the benefit, that the result can be compared to the original benchmark of the Tiny YOLOv2 model, which has been evaluated on the same data.[100] This comparison serves the purpose of measuring the loss of mAP that can be accounted to the pre-processing steps in the system or possible shortcomings of the Barracuda inference engine.

VOC2007 has a total of 4,952 test images with 12,032 objects and the corresponding annotations. Objects in the dataset belong to twenty different classes and the number of images per class ranges from 97 (sheep) to 2,007 (person). The task of the detection

[96] Cf. Gregor; Hevner (2013), p. 351.

[97] Cf. Gregor; Hevner (2013), p. 351.

[98] Cf. Peffers et al. (2008), p. 13.

[99] Cf. Everingham, Van Gool, et al. (2010), Lin, et al (2014).

[100] Cf. Redmon (2016).

challenge is described as follows: for each of the twenty classes, predict the bounding boxes of each object of that class in a test image (if any), with associated real-valued confidence. For a box to be considered a correct detection, the IoU must exceed a threshold of 50% (or 0.5). With a threshold that low the authors account for inaccuracies in bounding boxes in the ground truth data because the notion of a correct label (e.g., for a person with arms and legs spread) can be somewhat subjective. The model outputs that satisfy the IoU criterion are then assigned to the respective ground truth objects and ranked in terms of decreasing confidence. Multiple detections of the same object in an image were considered false detections.[101]

Table 9: Object classes and statistics on the VOC2007 test data.[102]

	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car
Images	204	239	282	172	212	174	721
Objects	285	337	459	263	469	213	1,201
	Cat	Chair	Cow	Dining table	Dog	Horse	Motorbike
Images	322	417	127	190	418	274	222
Objects	358	756	244	206	489	348	325
	Person	Potted plant	Sheep	Sofa	Train	TV/Monitor	**Total**
Images	2,007	224	97	223	259	229	**4,952**
Objects	4,528	480	242	239	282	308	**12,032**

7.1.2 Evaluation Metric

A common evaluation metric for object detectors is the mean Average Precision (mAP).[103] Because the class distribution in popular benchmark datasets is considerably non-uniform, a simple Accuracy measure (the ratio of correctly classified examples) is not considered a reliable metric. Instead, Average Precision (AP) was proposed as a single number metric that encapsulates Precision and Recall, both of which are established metrics for imbalanced datasets.[104]

In order to understand mAP, it is necessary to first introduce some more basic knowledge about predictions in the context of object detection. In the case of object detection, the model does not simply predict whether an object is in the image or not but rather where in the image the object is located. Since it is hard to predict exact locations, the bounding box predicted by the model (prediction) will most likely not have the same dimensions as the box in a labelled

[101] Cf. Everingham et al. (2015).

[102] Cf. Everingham et al. (2010), p. 310.

[103] Cf. Liu et al. (2021).

[104] Cf. Everingham et al. (2010).

test image (ground truth). In object detection a prediction is considered correct when three conditions are satisfied: the object is there, the model detects the object and the IoU is greater than a pre-defined threshold. This case is also called a True Positive (TP). In contrast, a False Positives (FP) exists when either the object is there, but the IoU is below the threshold, or when the object is not there but has been detected by the model. The last type of predictions is the False Negative (FN), that is an object is there but has not been detected. Since the model does not explicitly predict the absence of an object, True Negative (TN) results do not exist and are of no relevance to the metric. A visualisation of these cases can be found in Figure 23.

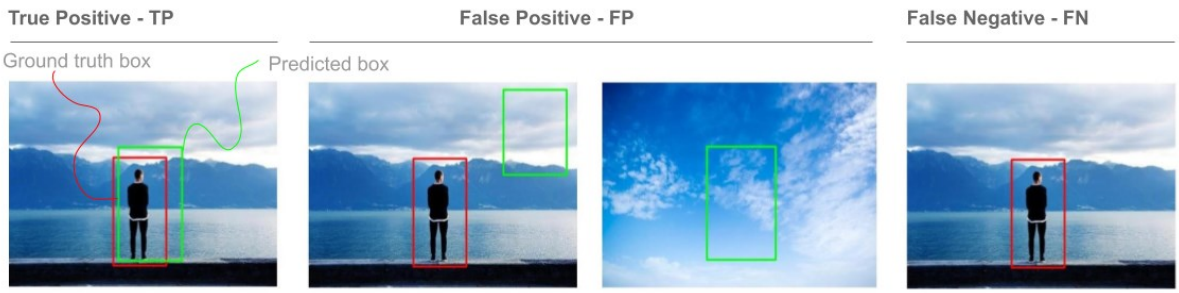


Figure 23: Types of predictions in object detection.[105]

Given the definition of TP, FP, and TN, it is possible to understand the two key metrics behind mAP. The first is called Precision and it measures the ratio of correct detections over all detected boxes ("What proportion of positive detections was actually correct?"). This means that for a Precision value of 0.5 the model is correct 50% of the time when it predicts that there is an object. The second is called Recall and measures the ratio of objects that have been correctly detected ("What proportion of actual positives was detected correctly?"). In this case a value of 0.5 can be interpreted in a way that the model predicts 50% of all objects correctly. The formulas for calculating Precision and Recall in terms of TP, FP, and TN look like follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

[105] Cf. El Aidouni (2019).

The higher the Precision, the more we can trust the model when it predicts an object (only few FP). The higher the Recall, the more objects the model detects (only few FN). A model performs best when both values are at 100%. However, Precision and Recall are not independent. Instead, Precision typically declines as Recall increases and vice versa.[106] This can be best observed when both values are plotted in a so-called Precision-Recall curve with Recall on the x-axis and Precision on the y-axis (cf. Figure 24). This curve allows to evaluate the whole model and compare it to others. The higher the curve, the better the model.

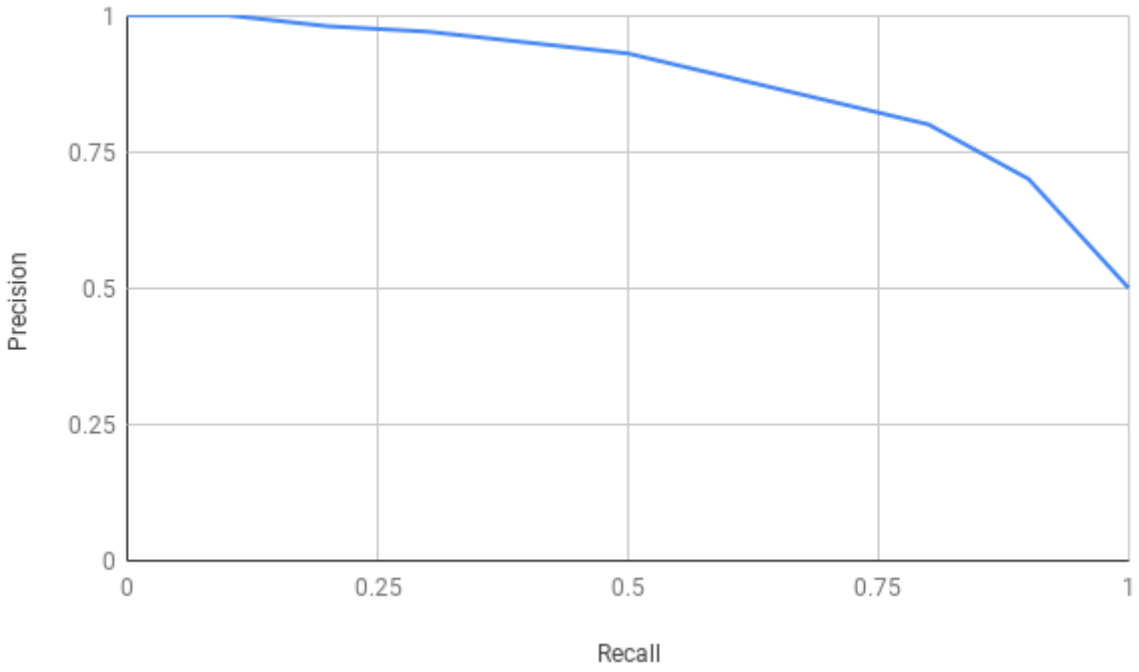


Figure 24: Example of a Precision-Recall Curve.[107]

Average Precision (AP) summarises the Precision-Recall curve by calculating the mean Precision across all Recall levels. This means that each Recall level r, the Precision p is interpolated by taking the maximum Precision measured at any point where the Recall $\tilde{r}$ exceeds r. In other words, find the maximum Precision value for a Recall higher than r for each of the Recall levels. Finally, average them out.

$$AP = \sum (r_n - r_{n-1}) p_{interp}(r), \quad where \quad p_{interp}(r) = \max_{\tilde{r}>r} p(\tilde{r}).$$

[106] Cf. Buckland; Gey (1994).

[107] Cf. El Aidouni (2019).

This also called all point AP in contrast to 11-point AP, where Precision was interpolated on eleven evenly spaced Recall values. Since 2010 however, PASCAL VOC stopped using 11-point AP in favour off all point mAP. Having defined the AP it is easy to derive the mAP. It averages the AP over all the classes in the dataset. Assuming the dataset has N classes, mAP is described by the following formula:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

7.1.3 Quantitative Test Set-Up

Because the model will be evaluated on a benchmark dataset, the camera module needs to be replaced by code that reads images from disc. Once read from disc, the images are resized directly, without any other transformations applied to them. This is a deviation from the original pre-processing procedure, where the images are first cropped to be of square dimensions and only then resized, keeping the aspect ratio. For the detection task mentioned earlier, where the goal is to detect all the objects correctly, the initial argument for cropping the image (cf. Section 6.2.1) no longer applies. The loss of information would be too big. The important fact for the evaluation is, that the detection quality of the system will not be enhanced by this but rather decline. Instead of working on a frame with its original aspect ratio, detection during evaluation is performed on distorted images. Thus, the evaluation results serve as a lower bound for the actual detection performance.

The resized images are then fed to the detection module, which works exactly like in the implementation. The model performs object detection and the output goes through a post-processing step. There, images are filtered with a confidence threshold of 0.005 and an IoU threshold of 0.45 during NMS. Contrary to the original implementation, at this point, there is no need for the visualisation of bounding boxes. Instead, the resulting bounding boxes are rescaled to match the dimensions of the input image and then saved in text files (Figure 25). For each class a single file is created where each line represents a detected object. The line has information about the image identifier, the confidence score and the coordinates of the bounding box given in left-top, right-bottom representation all separated by spaces. The procedure is implemented in Unity Editor version 2019.4.26f1 with Barracuda package version 1.0.4.

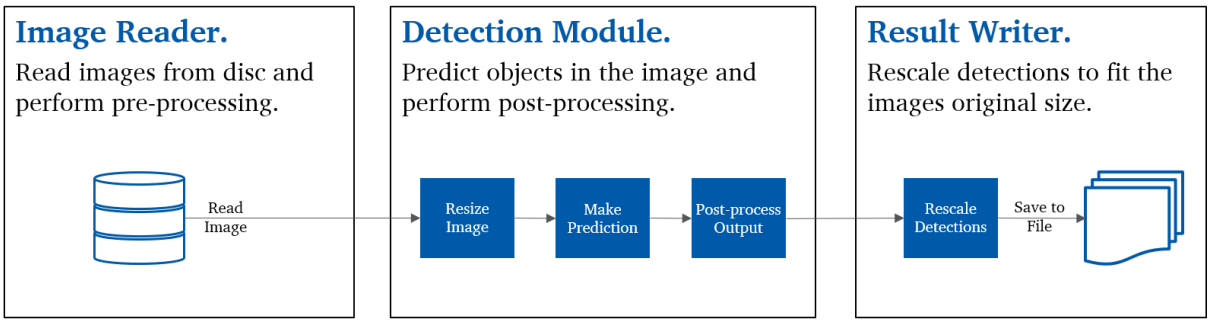


Figure 25: Evaluation Procedure Overview.

The final metric as described in Section 7.1.2 can be calculated using the VOC2012 Development Kit.[108] It is programmed in the Matlab desktop environment and only needs the ground truth information and the detection results from the previous step as input.[109] The ground truth information is available in the VOC2007 dataset as image annotations in xml-format. In addition, all results have been verified with another evaluation tool to rule out errors in handling the Development Kit.[110]

7.1.4 Results

Evaluated on the VOC2007 test data with the aforementioned parameters, the ML-based object detection system achieves a mAP of 55.04% over all classes. This is 2.07% lower than the value of 57.1% reported by the author of the original model implementation.[111] The AP for the classes ranges from 17.71% for 'bottle' to 72.23% for 'train'. An overview of the results is presented in Figure 26 and the Precision-Recall Curves for all the classes can be found in the Appendix. A detailed overview with the number of detected objects, TP, FP and AP are presented in Table 10.

[108] Cf. Everingham (2012).

[109] Cf. Mathworks (2021).

[110] Cf. Cartucho et al. (2018).

[111] Cf. Redmon (2016).

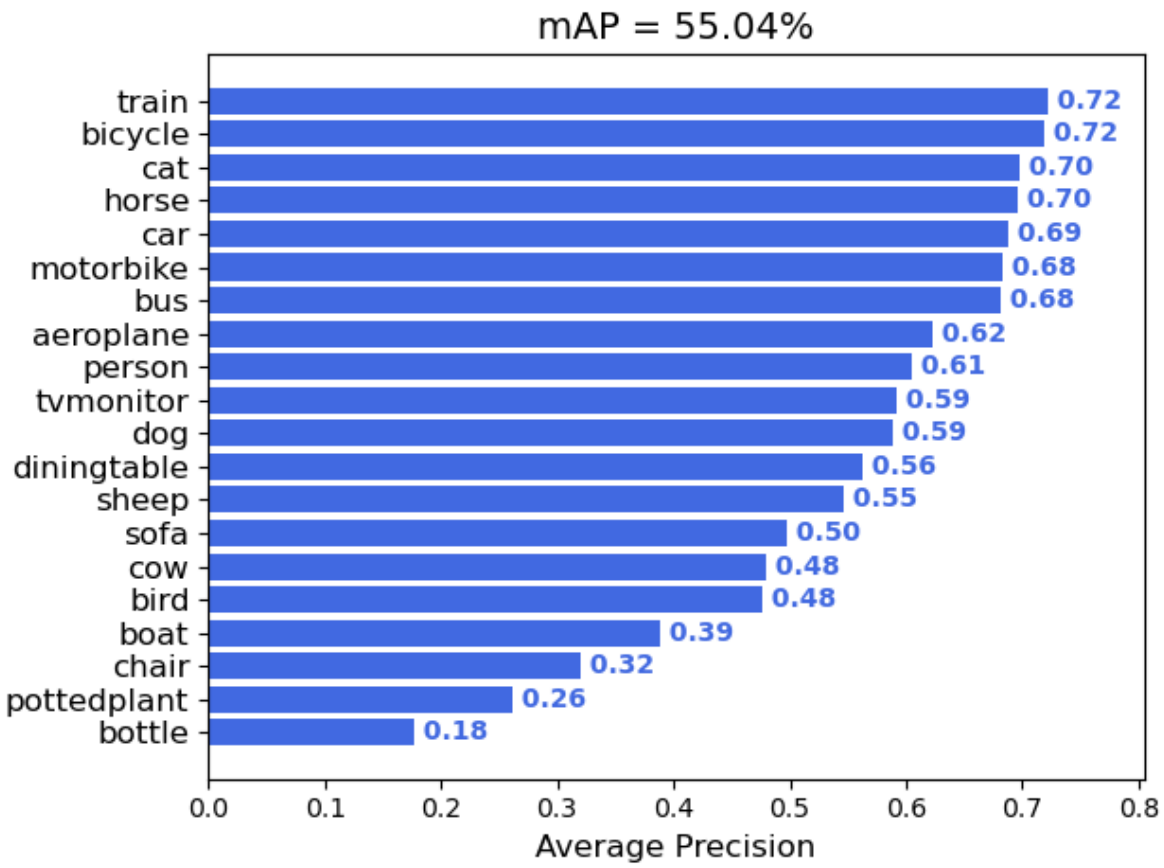


Figure 26: Evaluation Results: mAP and AP per Class.[112]

Table 10: Detailed Evaluation Results.

	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car
Detected	3,982	3,630	9,876	9,869	18,618	2,807	19,414
TP	222	291	325	185	209	171	987
FP	3,760	3,339	9,551	9,684	18,409	2,636	18,427
AP	62.29%	71.93%	47.69%	38.91%	17.71%	68.16%	68.83%
	Cat	Chair	Cow	Dining Table	Dog	Horse	Motorbike
Detected	2,809	31,531	3,389	3,897	4,229	2,248	2,274
TP	304	526	158	163	376	275	266
FP	2,505	31,005	3,231	3,734	3,853	1,973	2,008
AP	69.83%	32.07%	47.99%	56.23%	58.86%	69.63%	68.35%
	Person	Potted Plant	Sheep	Sofa	Train	TV/Monitor	Total
Detected	69,335	19,813	5,380	3,001	4,354	8,675	229,131
TP	3,557	313	178	189	251	245	9,191
FP	65,778	19,500	5,202	2,812	4,103	8,430	219,940
AP	60.51%	26.12%	54.60%	49.72%	72.23%	59.18%	55.04%

[112] Created with Cartucho et al. (2018).

7.2 Utility Assessment

7.2.1 Qualitative Research Interviews

Because it was not possible to conduct a broad user study, the utility of the artefact is evaluated with the help of industry experts.[113] Semi-structured interviews, also referred to as qualitative research interviews, are a way to gather valid and reliable data for an evaluative study.[114] The main goal of the interviews was to get an opinion from experts outside of university on the relevance of the technologies employed in the artefact for future work and to explore possible applications of the artefact as a solution to problems in industrial settings.

Three interviews were conducted over the span of one hour each with experts from the Energy, Defence, and Manufacturing sector. In order to capture as much information as possible, the interviews have been recorded and the respective transcripts were anonymised, as was agreed upon. Each participant received a description of the project in advance of the online meeting. Although each interview is different and develops its own dynamic, it was ensured that three thematic strands were covered. First, the interviewees were asked about their background and the amount of digitalisation and automation in their company. This way it was possible to get familiar with the projects that the interviewees deal and estimate their expertise on the matter. Second, the experts received a demonstration of a prototype of the ML-based object detection system for AR. With the proposed solution in mind, they then gave constructive feedback about the quality and potential use of the application in general. Lastly, the interviews evolved around future activities that the companies pursue in AR and ML so that the prototype could be mapped to possible domain specific applications.

7.2.2 Results

All three interviewees had a lot of expertise not only in their area of work but also concerning the challenges and possibilities of introducing new technologies in companies. Their companies want to or feel that they have to be more digital to be able to compete in the future. However, the financial resources that they allocate to innovative projects and AR

[113] This thesis was conducted during the Covid-19 pandemic. Despite an elaborated hygiene concept, it was not possible to meet the study participants due to government restrictions. In addition, the UK's withdrawal from the European Union made travel between England and Germany difficult.

[114] Cf. Saunders et al. (2015), pp. 388-430.

technology varies a lot. That the reason why acceptance was an important topic during the interviews. On the one hand, there was a consensus about the importance of collecting metrics and key performance indicators in order to get management approval for these kind of research projects. On the other hand, one of the interviewees highlighted the acceptance among the workforce. The usability of systems that have a lot of human-machine interaction should be assessed at an early stage of the project and the users should be included in the design process. The general notion of HMDs was that they are a good technology for the shop floor because they do not occlude the vision like, for example, screens, and workers need to be aware of their surroundings. One participant mentioned the security aspect. His company has hard times getting approval to use cloud environments because their products are subject to high safety regulations. HMDs that do not need to be connected to a network could the participants opinion be a good solution to still make use of ML models. However, the price for the equipment was perceived as being too high as to use them in the context of low price goods manufacturing. Further possible use cases that the interviewees mentioned are presented on the next page in Table 11. The table contains a short classification of the realm that the use case belongs to, as well as a more detailed description of the case.

Table 11. Possible use cases for ML-based AR systems that were collected from expert interviews.

Use case realm	Description
Context-aware work instructions	Detect the state of a product during assembly. Augment the product with a 3D overlay that describes the next assembly step. This way large manuals can be replaced. The time allocated to locating the manual, picking it up, finding the right page can be saved and helps workers to concentrate on the task at hand.
Part/Product Classification	Identify the type of a product correctly and fetch the corresponding information from a database. In environments with a big variety of products or configurations this helps to quickly get contextual information about the product, which is especially helpful for complex products such as circuit boards.
Workforce enablement	If workers can get a second opinion on the spot, it enables companies to allocate less trained workers to more complex inspection tasks. Training itself is very expensive and takes a long time.
Goods-in Quality Inspection	For incoming goods there is no standardised or automated inspection process in place. A mobile inspection solution that incorporates intelligence trough ML models can speed things up significantly and quickly decide which goods should be reclaimed.
Maintenance, Repair, Overhaul	Identify damaged parts in complex and high costs products.
Documentation	Automate the documentation of a process. By automatically saving information about the steps that were performed, for example during assembly, companies have a more reliable way to document their processes. At the same time the most recent data of the process is available via a connection to the database in contrast to paper manuals that may be outdated.
Fault Detection	Detect errors on soldered components and collect data about where and how often these errors occur. The data can be used in a root cause analysis that may lead to improved production processes.

8 Discussion

The implementation of the ML-based object detection system for HoloLens led to several findings regarding the maturity of the technology and challenges of implementing such a system. Furthermore, a variety of current and potential use cases for ML-based AR systems have been identified.

The evaluation on the VOC2007 object detection benchmark addressed RQ 2, asking how well the ML models performs on a HoloLens. First and foremost it revealed that neural networks can achieve the same accuracy when deployed on an HMD as they would on conventional hardware. As was to be expected, however, they could not achieve a high performance in terms of speed. With regards to RQ 1 and the challenges of implementing a ML-based system for HoloLens, the requirement analysis provided a comprehensive discussion and overview of the technical possibilities. It became evident that despite Barracuda provides a simple way to run ML models on the HoloLens by easily integrating it in the Unity ecosystem. On the downside, like other inference engines, it does not support every model architecture, yet. Furthermore, the resulting implementation can serve as a blueprint for applications that want to use the state-of-the-art of ML on HMDs. Nonetheless, there are limitations due to the scope of this thesis and a lot of potential for future research remains. The literature review on ML-based AR systems revealed current use cases for the combination of the two technologies, from which four criteria for a classification framework could be derived. Furthermore, the expert interviews provided feedback on the potential of the ML-based object detection system in particular to be adaptable to future use cases. The following paragraphs discuss the findings, limitations and the potential for future research with respect to the RQs and the current state of knowledge.

One main aspect that could be shown, that running a ML-based object detection model with the frameworks available for HoloLens barely affects the models' ability to detect objects. With a mAP of 55.04% the detection quality is only slightly worse than the official benchmark with a 57.1% mAP.[115] The missing 2.07% can most likely be attributed to other circumstances like the training process or the conversion from PyTorch to ONNX format. Accordingly, it is safe to assume that in terms of detection quality the system works as well on the HoloLens as it does on conventional hardware. To the best of the authors' knowledge, this has been the first benchmark test confirming that ML models can achieve similar results on the inferior

[115] Cf. Redmon (2016).

hardware of HMDs and do not suffer from a decline in accuracy. For example, it supports the validity of the results that von Atzigen et al. (2021) reported for their screw head detection task, which was evaluated on a small custom dataset and would not be that meaningful without any reference validating the robustness of the ML model that was deployed. Furthermore, the results implicate that ML models performing other tasks than the detection of common objects can also be run on an HMD without forfeiting accuracy, which is an important metric if a developer wants to get management approval for this kind of project.

Even though the accuracy of the system is good, running it on the HoloLens brings severe losses in speed. In line with what Atzigen et al. (2021) report, the ML model needs roughly one second to process a single frame. This is significantly slower than the 207 fps from the original benchmark.[116] Nevertheless, for many real-world applications the computational power should still be sufficient. Apart from that, it was shown that ML can be deployed to an HMD without dependencies to external infrastructure like WLAN or the need for computational resources like cloud or edge servers. In contrast to these streaming solutions, the on-device approach gives users the benefits of maximum mobility, increased data privacy, and immediate responses. The hardware disadvantage of HMDs, on the other hand, will eventually become smaller and smaller.

Another contribution of this thesis lies in the thorough analysis of technical requirements and the breakdown of the system into separate modules. Each module serves a specific purpose and can be adjusted to different conditions. For example, in order to deploy the system on another HMD it is only necessary to change how the Camera Module accesses the HMDs visual sensors. Similarly, the Detection Module can be adjusted to use another ML model that detects different objects. In both cases the other parts of the system can remain in operation as they are. This way, the implementation can be used as a blueprint for future applications. By spending less time on the development of a prototype, researchers can iterate faster and focus on the results and evaluations of their projects, instead of putting too much effort into the details of an implementation.

With respect to future applications it is also noteworthy that at the time of completing this work, the Barracuda inference engine only supports a limited number of neural network architectures. In the last months, however, it has shown great progress, including a growing community and is close to a new major release. Accordingly, there will be more sophisticated ML models available for inference on HMDs, enabling this technology to serve evermore use

[116] Cf. Redmon (2016).

cases. The models may, for example, replace some of the possibly inferior approaches in the AR applications mentioned in Section 2.1.1. The same applies to some of the ML models from Section 2.2, which could also be deployed on an HMD. The ML-based object detection that was implemented here can by now be improved by switching from Tiny YOLOv2 to TinyYOLOv3. Furthermore, the utility assessment revealed the potential for the system to be used or adapted to multiple use cases, especially in the context of product detection and quality inspection.

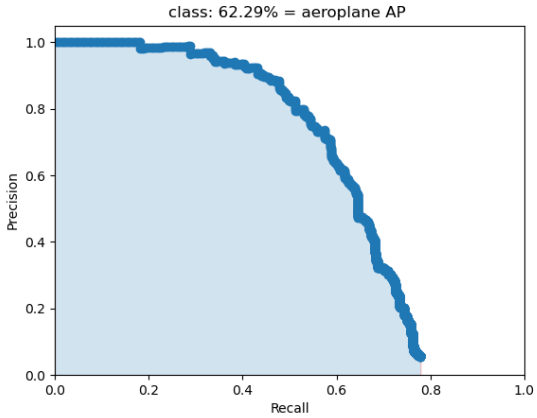
Apart from the technical performance and the adaption to new use cases, there remains the question of industry-readiness. Evaluating the acceptance of the system among industry workers lay outside the scope of this thesis. During the interviews it became evident that it is of utmost importance to include users in the design process. Consequently, as a next step, it would be valuable to gain additional insights on how the system might be used in practice to infer important knowledge on how to design these types of systems for optimal user friendliness. This includes investigations on how holograms are perceived in a work environment, e.g. with respect to accurate placement of holographic information or the possibility of motion sickness. Furthermore, the system for ML-based object detection includes a minimum of operability. For the future it would not only be interesting to study if objects could be detected, but also to see how the user interacts with the hologram placed over the object.

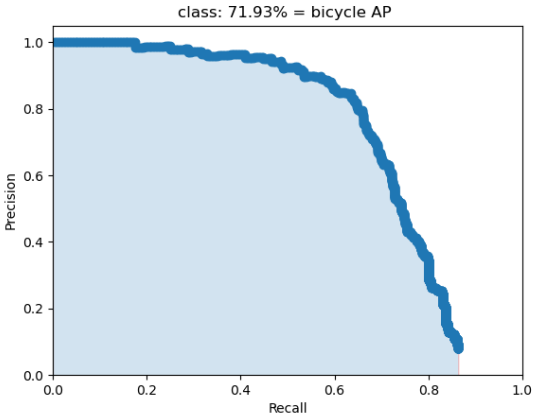
For now the HMD has been used as a means to access the camera, to execute the detection model and to visualise the results. However, Li et al. (2020) have shown that including information from other sensors can improve results even further. Other than simply adapting existing ML use cases to HMDs, possibly the most interesting area for future research lies in even improving these cases through the use of additional sensoric data. This can not only generate better results for known applications, but may also lead the way to many new and creative solutions that exhaust the synergies of ML and HMDs.

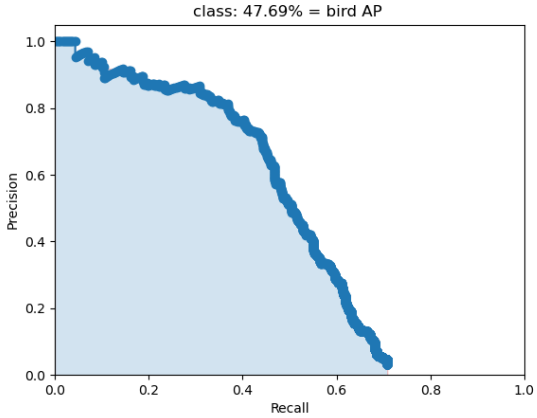
9 Conclusion

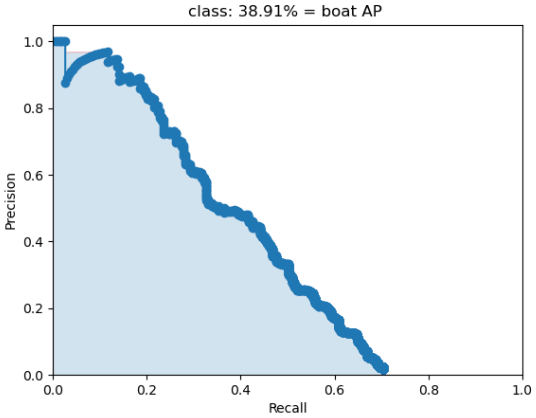
At the core of this thesis is the development of a ML based object detection system for augmented reality. Unlike in most of these systems the ML is executed on an HMD with relatively low computational capabilities. The contribution of this work regarding potential applications in industrial quality inspection and beyond is twofold. Firstly, the implementation comes with a rigor requirement analysis that shows the current possibilities of accessing the HMD sensors, of running a ML model on the device and of visualising the immediate results. The modular design of the object detection system allows other researchers and software developers to quickly adapt it to other use cases. The blueprint nature of the design enables quick development cycles via the exchange of single parts of the system without the need for change on other parts. Secondly, the evaluation on an established benchmark data set was the first time a system that is only using technological frameworks which are fit for AR has been proven to achieve the same accuracy as those ML models that are backed up by established frameworks. Therefore, ML models can be integrated in that kind of system more safely. Researchers can focus on the evaluation of their models on custom-built data sets that are relevant to their actual use case. The robust accuracy performance of well-known model architectures is now implicit and does not need to be proven again for their use with inference engines in AR. The speed of the inference on the other hand depends heavily on the computational capabilities of the HMD. Overall the findings indicate that standalone ML based AR systems show great potential for industrial use. With further advancements in HMD technology and the respective ML engines the possibilities will only become more manifold.

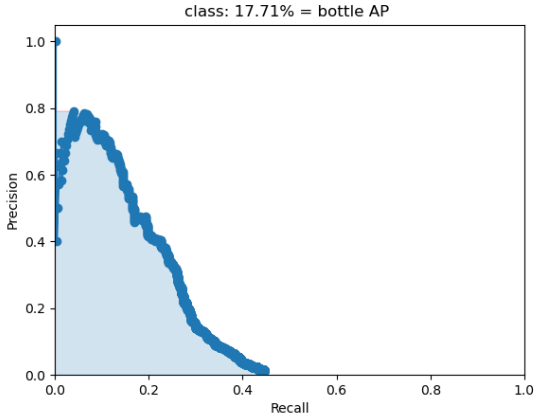
Appendix

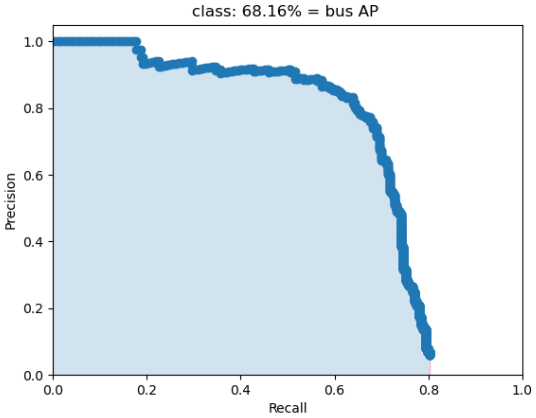

class: 62.29% = aeroplane AP
Precision
Recall

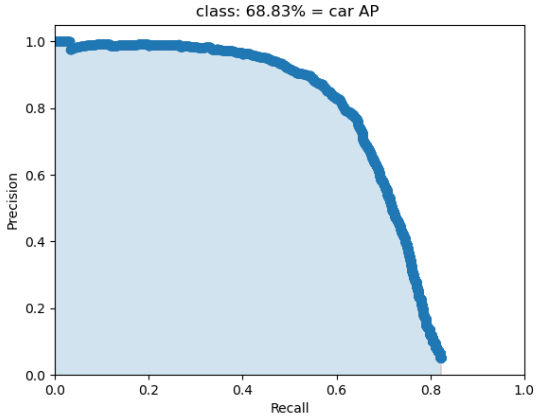

class: 71.93% = bicycle AP
Precision
Recall

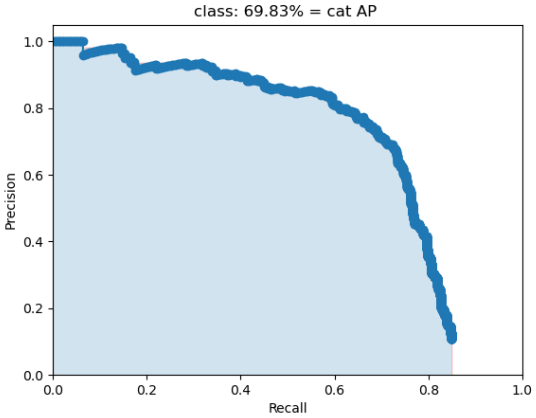

class: 47.69% = bird AP
Precision
Recall

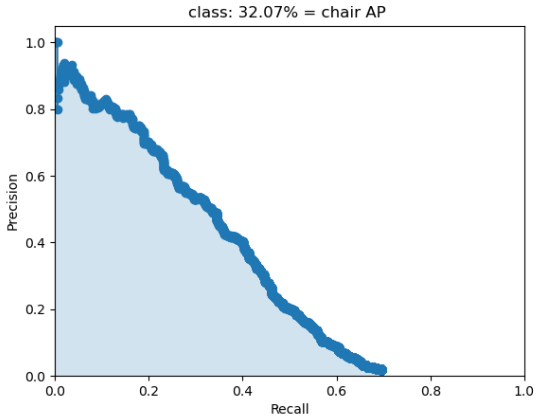

class: 38.91% = boat AP
Precision
Recall

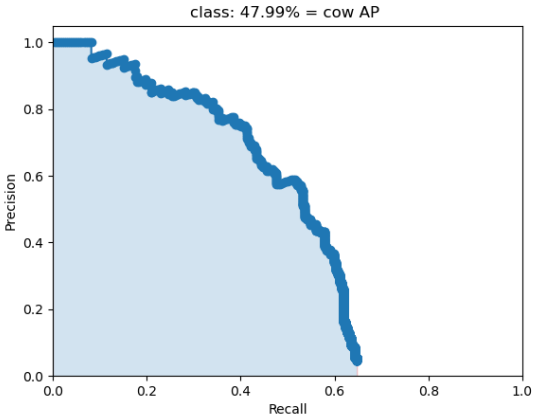

class: 17.71% = bottle AP
Precision
Recall

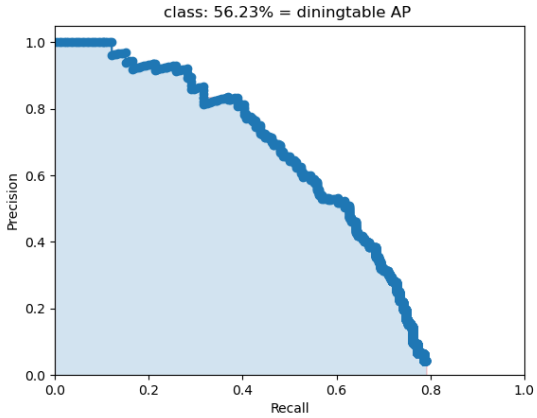

class: 68.16% = bus AP
Precision
Recall

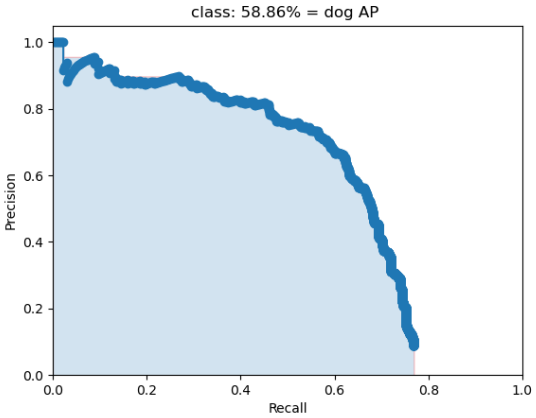

class: 68.83% = car AP
Precision
Recall

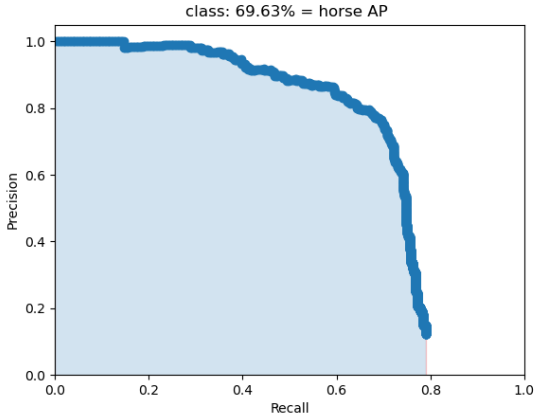

class: 69.83% = cat AP
Precision
Recall

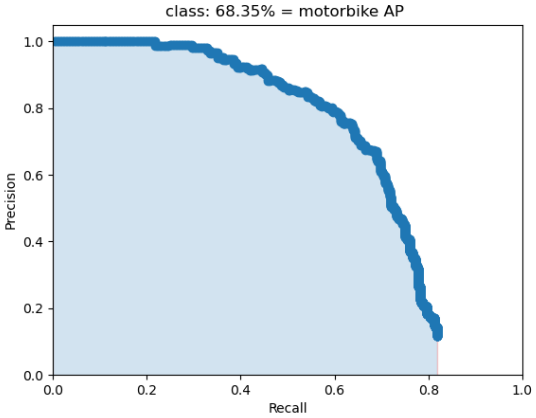

class: 32.07% = chair AP
Precision
Recall

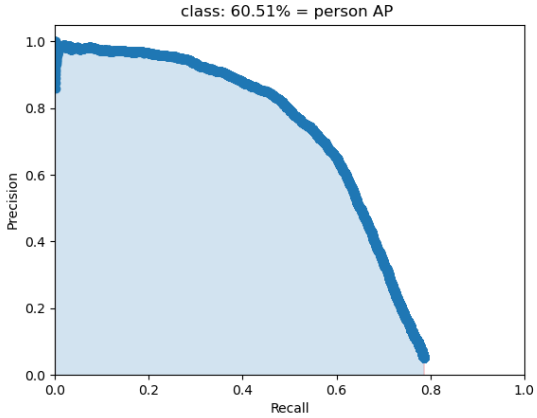

class: 47.99% = cow AP
Precision
Recall

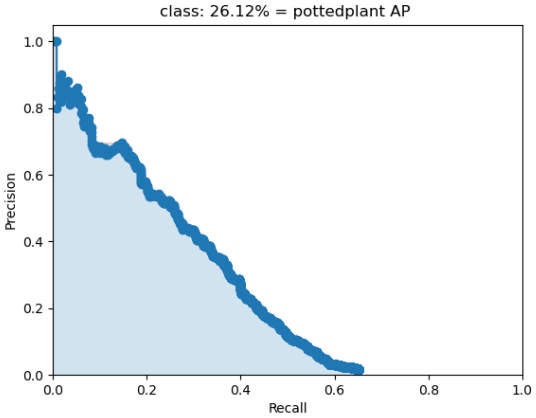

class: 56.23% = diningtable AP
Precision
Recall

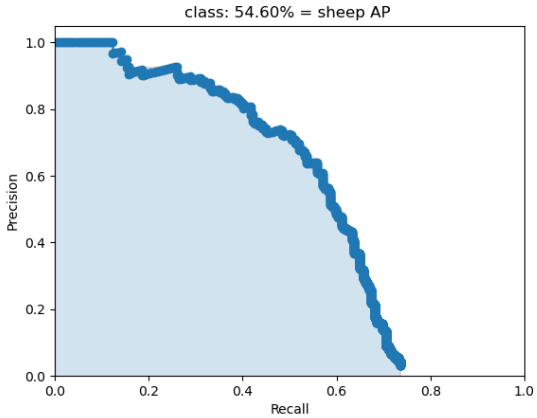

class: 58.86% = dog AP
Precision
Recall

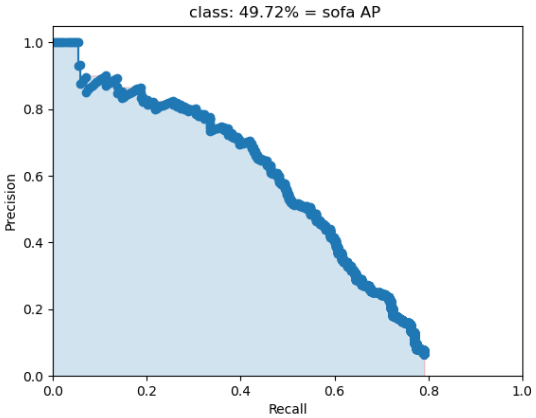

class: 69.63% = horse AP
Precision
Recall

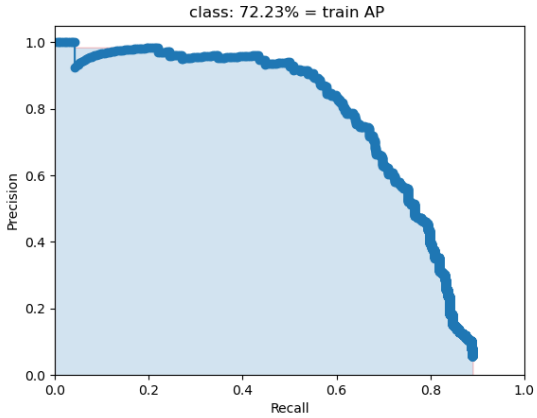

class: 68.35% = motorbike AP
Precision
Recall

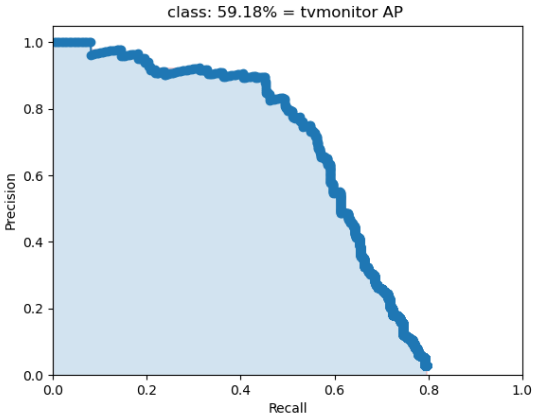

class: 60.51% = person AP
Precision
Recall


class: 26.12% = pottedplant AP
Precision
Recall


class: 54.60% = sheep AP
Precision
Recall


class: 49.72% = sofa AP
Precision
Recall


class: 72.23% = train AP
Precision
Recall


class: 59.18% = tvmonitor AP
Precision
Recall

References

Angrisani, Leopoldo; Arpaia, Pasquale; Esposito, Antonio; Moccaldi, Nicola (2020): A Wearable Brain–Computer Interface Instrument for Augmented Reality-Based Inspection in Industry 4.0. In: IEEE Transactions on Instrumentation and Measurement, 4, pp. 1530–1539.

Ashikhmin, Andrey (2020): TFClassify-Unity-Barracuda. https://github.com/Syn-McJ/TFClassify-Unity-Barracuda/blob/master/Assets/Scripts/TextureTools.cs, Accessed April 25, 2021.

Bahri, Haythem; Krcmarik, David; Koci, Jan (2019): Accurate Object Detection System on HoloLens Using YOLO Algorithm. In: 2019 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), pp. 219–224.

Bochkovskiy, Alexey; Wang, Chien-Yao; Mark Liao, Hong-Yuan (2020): YOLOv4: Optimal Speed and Accuracy of Object Detection. http://arxiv.org/pdf/2004.10934v1. Accessed April 25, 2021.

Bruder, Gerd; Steinicke, Frank; Valkov, Dimitar; Hinrichs, Klaus (2010): Augmented Virtual Studio for Architectural Exploration. In: Proceedings of Virtual Reality International Conference (VRIC), 2010.

Buckland, Michael; Gey, Fredric (1994): The relationship between Recall and Precision. In: Journal of the American Society for Information Science, pp. 12–19.

Cartucho, Joao; Ventura, Rodrigo; Veloso, Manuela (2018): Robust Object Recognition Through Symbiotic Deep Learning In Mobile Robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 2336-2341.

Chang, M. M. L.; Ong, S. K.; Nee, A. Y. C (2017): AR-guided Product Disassembly for Maintenance and Remanufacturing. In: *Procedia CIRP*, pp. 299–304.

Chen, Bobby (2019): Two-stage vs One-stage Detectors. https://github.com/yehengchen/Object-Detection-and-Tracking/blob/master/Two-stage%20vs%20One-stage%20Detectors.md. Accessed May 23, 2021.

Common Visual Data Foundation (2020): *Open Images Dataset V6 + Extensions.* https://storage.googleapis.com/openimages/web/index.html. Accessed April 18, 2021.

Coppens, Adrien (2017): Merging real and virtual worlds: An analysis of the state of the art and practical evaluation of Microsoft Hololens. Master Thesis, Faculty of Sciences, University of Mons, Belgium.

Damiani, Lorenzo; Demartini, Melissa; Guizzi, Guido; Revetria, Roberto; Tonelli, Flavio (2018): Augmented and virtual reality applications in industrial systems: A qualitative review towards the industry 4.0 era. In: International Federation of Automatic Control (IFAC), 11, pp. 624–630.

Dasgupta, Archi; Manuel, Mark; Mansur, Rifat Sabbir (2020): Towards Real Time Object Recognition For Context Awareness in Mixed Reality: A Machine Learning Approach. In: IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2020, pp. 262–268.

Del Amo, Iñigo Fernández; Galeotti, Elisa; Palmarini, Riccardo; Dini, Gino; Erkoyuncu, John; Roy, Rajkumar (2018): An innovative user-centred support tool for Augmented Reality maintenance systems design: a preliminary study. In: Procedia CIRP, pp. 362–367.

Dhiman, Hitesh; Martinez, Sascha; Paelke, Volker; Röcker, Carsten (2018): Head-Mounted Displays in Industrial AR-Applications: Ready for Prime Time? In: HCI in Business, Government, and Organizations, pp. 67-78.

Eckert, Martin; Blex, Matthhias; Friedrich, Christoph M. (2018): Object Detection Featuring 3D Audio Localization for Microsoft HoloLens - A Deep Learning based Sensor Substitution Approach for the Blind. In: Proceedings of the 11th Intternational Joint Conference on Biomedical Engineering Systems and Technologies, pp. 555–561.

Edwards-Steward, Amanda; Hoyt, Tim; Reger, Greg M. (2016): Classifying different types of augmented reality technology. In: Annual Review of CyberTherapy and Telemedicine, pp. 199-202.

Egger, Johannes; Masood, Tariq (2020): Augmented reality in support of intelligent manufacturing – A systematic literature review. In: Computers & Industrial Engineering, pp. 106-195.

El Aidouni, Manal (2019): Evaluating Object Detection Models: Guide to Performance Metrics. https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html. Accessed May 15, 2021.

Evans, Gabriel; Miller, Jack; Iglesias Pena, Mariangely; MacAllister, Anastacia; Winer, Eliot (2017): Evaluating the Microsoft HoloLens through an augmented reality assembly application. In: Degraded Environments.

Everingham, Mark (2007): The PASCAL Visual Object Classes Challenge 2007. http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html. Accessed May 23, 2021.

Everingham, Mark (2012): The PASCAL Visual Object Classes Challenge 2012. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html. Accessed May 23, 2021.

Everingham, Mark; van Gool, Luc; Williams, Christopher; Winn, John; Zisserman, Andrew (2010): The PASCAL Visual Object Classes (VOC) Challenge. In: International Journal of Computer Vision, pp. 303-338.

Everingham, Mark; Eslami, Ali; van Gool, Luc; Williams, Christopher; Winn, John; Zisserman, Andrew (2015): The Pascal Visual Object Classes Challenge: A Retrospective. In: International Journal of Computer Vision, pp. 98–136.

Farasin, Alessandro; Peciarolo, Francesco; Grangetto, Marco; Gianaria, Elena; Garza, Paolo (2020): Real-time Object Detection and Tracking in Mixed Reality using Microsoft HoloLens. In: Proceedings of the 15th International Joint Conference on Computer Vision, pp. 165–172.

Freeman, Justin (2020): Content enhancement with augmented reality and machine learning. In: Journal of Southern Hemisphere Earth Systems Science, pp. 143-150.

Govindarajan, Usharani Hareesh; Trappey, Amy J. C.; Trappey, Charles V. (2018): Immersive Technology for Human-Centric Cyberphysical Systems in Complex Manufacturing Processes: A Comprehensive Overview of the Global Patent Profile Using Collective Intelligence. In: Complexity, pp. 1–17.

Gregor, Shirley; Hevner, Alan (2013): Positioning and Presenting Design Science Research for Maximum Impact. In: MIS Quarterly, pp. 337-355.

Henderson, Steven; Feiner, Steven (2011): Exploring the benefits of augmented reality documentation for maintenance and repair. In: IEEE transactions on visualization and computer graphics, pp. 1355–1368.

Hevner, Alan R.; March, Salvatore T.; Park, Jinsoo (2004): Design Science in Information Systems Research. In: MIS Quarterly, pp. 75-105.

Ibáñez, María-Blanca; Delgado-Kloos, Carlos (2018): Augmented reality for STEM learning: A systematic review. In: Computers & Education, pp. 109–123.

Jaccard, Paul (1901): Etude de la distribution florale dans une portion des Alpes et du Jura. In: Bulletin de la Societe Vaudoise des Sciences Naturelles, pp. 547-579.

Kopardekar, Parimal; Mital, Anil; Anand, Sam (1993): "Manual, Hybrid and Automated Inspection Literature and Current Research. In: Integrated Manufacturing Systems, pp. 18-29.

Krenzer, Adrian; Stein, Nikolai; Griebel, Matthias; Flath, Christoph M. (2019): Augmented Intelligence for Quality Control of Manual Assembly Processes using Industrial Wearable Systems. In: 40th International Conference on Information *Systems, 2019, pp.* 4724-4732.

Li, Xiang; Tian, Yuan; Zhang, Fuyao; Quan, Shuxue; Xu, Yi (2020): Object Detection in the Context of Mobile Augmented Reality. In: IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 2020, pp. 156–163.

Liao, Zhibin; Carneiro, Gustavo (2016): On the importance of normalisation layers in deep learning with piecewise linear activation units. In: IEEE Winter Conference on Applications of Computer Vision (WACV), 2019, pp. 1-8.

Lin, Tsung-Yi; Maire, Michael; Belongie, Serge; Hays, James; Perona, Pietro ; Ramanan, Deva; Dollár, Piotr; Zitnick, Lawrence (2014): Microsoft COCO: Common Objects in Context. In: Computer Vision – ECCV, *2014*, pp. 740–755.

Lin, Tsung-Yi; Goyal, Priya; Girshick, Ross ; He, Kaiming; Dollár, Piotr (2017): Focal Loss for Dense Object Detection. In: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2999-3007.

Liu, Luyang; Li, Hongyu; Gruteser, Marco (2019): Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In: The 25th Annual International Conference on Mobile Computing and Networking, pp. 1–16.

Liu, Wei; Anguelov, Dragomir; Erhan, Dumitru; Szegedy, Christian (2016): SSD: Single Shot MultiBox Detector." In: Computer Vision – ECCV, 2016, pp. 21-37.

Liu, Yang; Sun, Peng ; Wergeles, Nickolas; Shang, Yi (2021): A survey and performance evaluation of deep learning methods for small object detection. In: Expert Systems with Applications.

Mascareñas, David D. L.; Ballor, Jo Ann P.; McClain, Oscar L.; Mellor, Miranda A.; Shen, Chih-Yu; Bleck, Brian; Morales, John (2020): Augmented Reality for Next Generation Infrastructure Inspections. In: Structural Health Monitoring.

Mathworks (2021): Matlab. https://www.mathworks.com/products/matlab.html. Accessed May 22, 2021.

Microsoft (2019): Mixed Reality-Documentation: Locatable camera. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/locatable-camera. Accessed April 25, 2021.

Microsoft (2020a): UWP Documentation: What's a Universal Windows Platform (UWP) app? https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide. Accessed May 23, 2021.

Microsoft (2020b): HoloLensForCV GitHub. https://github.com/Microsoft/HoloLensForCV. Accessed March 27, 2021.

Microsoft (2020c): Windows AI Documentation: Windows Machine Learning. https://docs.microsoft.com/en-us/windows/ai/windows-ml/. Accessed May 23, 2021.

Microsoft (2020d): Dynamics 365 Documentation: Authoring Gestures. https://docs.microsoft.com/en-us/dynamics365/mixed-reality/guides/authoring-gestures-hl2. Accessed May 23, 2021.

Microsoft (2020e): Mixed Reality-Documentation: Hologram stability. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/hologram-stability#hologram-render-distances. Accessed April 25, 2021.

Microsoft (2021a): HoloLens 2 Tech Specifications. https://www.microsoft.com/en-us/p/hololens-2/91pnzzznzwcp?activetab=pivot:techspecstab. Accessed April 13, 2021.

Microsoft (2021b): Mixed Reality-Documentation: Camera setup in Unity. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/camera-in-unity#clip-planes. Accessed April 25, 2021.

Milgram, Paul; Kishino, Fumio (1994): A Taxonomy of Mixed Reality. In: IEICE Transactions on Information Systems, http://web.cs.wpi.edu/~gogo/courses/cs525H_2010f/papers/Milgram_IEICE_1994.pdf. Accessed March 28, 2021.

Milgram, Paul; Takemura, Haruo; Utsumi, Akira; Kishino, Fumio (1994): Augmented reality: a class of displays on the reality-virtuality continuum. In: Proceedings of SPIE - The International Society for Optical Engineering, pp. 282–292.

Miller, Jack; Hoover, Melynda; Winer, Eliot (2020): Mitigation of the Microsoft HoloLens' hardware limitations for a controlled product assembly process. In: The International Journal of Advanced Manufacturing Technology, pp. 1741–1754.

Mital, Anil; Govindaraju, M.; Subramani, B. (1998): A comparison between manual and hybrid methods in parts inspection. In: Integrated Manufacturing Systems, pp. 344-349.

Open Neural Network Exchange (2021a): *ONNX Model Zoo GitHub.* https://github.com/onnx/models#object-detection--image-segmentation-. . Accessed May 17, 2021.

Open Neural Network Exchange (2021b): *tf2onnx GitHub.* https://github.com/onnx/tensorflow-onnx. Accessed May 23, 2021.

Palmarini, Riccardo; Erkoyuncu, John Ahmet; Roy, Rajkumar; Torabmostaedi, Hosein (2018): A systematic review of augmented reality applications in maintenance. In: Robotics and Computer-Integrated Manufacturing, pp. 215–228.

Peffers, Ken; Tuunanen, Tuure; Rothenberger, Marcus A.; Chatterjee, Samir (2008): A Design Science Research Methodology for Information Systems Research. In: Journal of Management Information Systems, pp. 45-77.

Pullan, Graham; Chuan, Tan; Wong, Daren; Jasik, Franek (2019): Enhancing Web-Based CFD Post-Processing using Machine Learning and Augmented Reality. In: AIAA Scitech 2019 Forum.

PyTorch (2019): *Documentation.* https://pytorch.org/docs/stable/onnx.html?highlight=onnx#module-torch.onnx. Accessed May 23, 2021.

Qiu, Heqian; Li, Hongliang; Wu, Qingbo; Meng, Fanman; Ngi Ngan, King; Shi, Hengcan (2019): A2RMNet: Adaptively Aspect Ratio Multi-Scale Network for Object Detection in Remote Sensing Images. In: Remote Sensing, pp. 1594-1616.

Redmon, Joseph. (2016): YOLO: Real-Time Object Detection. https://pjreddie.com/darknet/yolov2/. Accessed April 13, 2021.

Redmon, Joseph; Farhadi, Ali (2017): YOLO9000: Better, Faster, Stronger. In: Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, pp. 6517-6525.

Redmon, Joseph; Farhadi, Ali (2018): "YOLOv3: An Incremental Improvement." http://arxiv.org/pdf/1804.02767v1. Accessed April 13, 2021.

Redmon, Joseph; Divvala, Santosh; Girshick, Ross; Farhadi, Ali (2016): You Only Look Once: Unified, Real-Time Object Detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788.

Ren, Shaoqing; He, Kaiming; Girshick, Ross; Sun, Jian (2015): Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Advances In Neural Information Processing, pp. 91-99.

Ren, Shaoqing; He, Kaiming; Girshick, Ross; Sun, Jian (2017): Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1137-1149.

Roth, Elisa; Möncks, Mirco; Bohné, Thomas; Pumplun, Luisa (2020): Context-Aware Cyber-Physical Assistance Systems in Industrial Systems: A Human Activity Recognition Approach. In: 2020 IEEE International Conference on Human-Machine Systems (ICHMS), pp. 1-6.

Saunders, M. N. K.; Lewis, Philip; Thornhill, Adrian (2015): Research methods for business students. 7th Edition. New York: Pearson Education.

Schmitt, Jacqueline; Bönig, Jochen; Borggräfe, Thorbjörn; Beitinger, Gunter; Deuse, Jochen (2020): Predictive model-based quality inspection using Machine Learning and Edge Cloud Computing. In: Advanced Engineering Informatics, pp. 101101.

Sein, Henfridsson; Rossi, Purao (2011): Action Design Research. In: MIS Quarterly, pp. 45–77.

Shen, Jinyang; Dong, Zhanxun; Qin, Difu; Lin, Jingyu; Li, Yahong (2020): iVision: An Assistive System for the Blind Based on Augmented Reality and Machine Learning. In: Universal Access in Human-Computer Interaction. Design Approaches and Supporting Technologies, pp. 393–403.

Singh, Bharat; Davis, Larry S. (2018): An Analysis of Scale Invariance in Object Detection SNIP. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3578-3587.

Sinha, Rajat Kumar; Pandey, Ruchi; Pattnaik, Rohan (2017): Deep Learning For Computer Vision Tasks: A review. https://arxiv.org/abs/1804.03928. Accessed January 20, 2021.

Su, Yongzhi; Rambach, Jason; Minaskan, Nareg; Lesur, Paul; Pagani, Alain; Stricker, Didier (2019): Deep Multi-state Object Pose Estimation for Augmented Reality Assembly. In: IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), 2019, pp. 222-227.

Svensson, Jan; Atles, Jonatan (2018): Object Detection in Augmented Reality. Master's Theses in Mathematical Sciences FMAM05 2018. http://lup.lub.lu.se/student-papers/record/8964243. Accessed December 8, 2020.

Szajna, Andrzej; Szajna, Janusz; Stryjski, Roman; Sąsiadek, Michał; (2019): The Application of Augmented Reality Technology in the Production Processes. In Intelligent Systems in Production Engineering, pp. 316–324.

Szajna, Andrzej; Stryjski, Roman; Woźniak, Waldemar; Chamier-Gliszczyński, Norbert; Kostrzewski, Mariusz (2020): Assessment of Augmented Reality in Manual Wiring Production Process with Use of Mobile AR Glasses. In: Sensors.

Tan, Mingxing; Pang, Ruoming; Le V., Quoc (2020): EfficientDet: Scalable and Efficient Object Detection. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 10778–10787.

Tiwari, Mukesh; Sinhai, Rakesh (2017): A Review of Detection and Tracking of Object from Image and Video Sequences. In: International Journal of Computational Intelligence Research, pp. 745-765.

Trestioreanu, Lucian; Glauner, Patrick; Meira, Jorge Augusto; Gindt, Max; State, Radu (2020): Using Augmented Reality and Machine Learning in Radiology. In: Innovative Technologies for Market Leadership, pp. 89–106.

Unity Technologies (2020): Unity uGUI Documentation. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/HOWTO-UIWorldSpace.html. Accessed April 25, 2021.

Unity Technologies (2021a): Unity Homepage. https://unity.com/. Accessed April 13, 2021.

Unity Technologies (2021b): Unity v2019.4 Manual. https://docs.unity3d.com/2019.4/Documentation/Manual/index.html. Accessed April 25, 2021.

Unity Technologies (2021c): Unity v2019.4 Script Reference. https://docs.unity3d.com/2019.4/Documentation/ScriptReference/. Accessed April 23, 2021.

Unity Technologies (2021d): *Unity Barracuda Manual.* https://docs.unity3d.com/Packages/com.unity.barracuda@1.4/manual/index.html. Accessed April 25, 2021.

Unity Technologies (2021e): *Unity v2020.3 Manual.* https://docs.unity3d.com/Manual/index.html. Accessed May 23, 2021.

Vávra, P.; Roman, J.; Zonča, P.; Ihnát, P.; Němec, M.; Kumar, J.; Habib, N.; and El-Gendi, A. (2017): Recent Development of Augmented Reality in Surgery: A Review. In: Journal of Healthcare Engineering.

Venable, John; Pries-Heje, Jan; Baskerville, Richard (2016): FEDS: a Framework for Evaluation in Design Science Research. In: European Journal of Information Systems, pp. 77–89.

von Atzigen, Marco; Liebmann, Florentin; Hoch, Armando; Bauer, David E. ; Snedeker, Jess Gerrit; Farshad, Mazda; Fürnstahl, Philipp (2021): HoloYolo: A proof-of-concept study for marker-less surgical navigation of spinal rod implants with augmented reality and on-device machine learning. In: The International Journal of Medical Robotics and Computer Assisted Surgery.

Vulcan Technologies (2018): HoloLensCameraStream GitHub. https://github.com/VulcanTechnologies/HoloLensCameraStream. Accessed March 27, 2021.

Wortmann, Henrik (2020): Objekterkennung unter Nutzung von Machine Learning für Augmented Reality Anwendungen. Faculty of Engineering and Comuter Science, Hamburg University of Applied Sciences.

Wu, Yuxin; Kirillov, Alexander; Massa, Francisco (2019): Detectron2. https://github.com/facebookresearch/detectron2. Accessed February 19, 2021.

Yew, A.W.W.; Ong, S. K.; Nee, A.Y.C. (2016): Towards a griddable distributed manufacturing system with augmented reality interfaces. In: Robotics and Computer-Integrated Manufacturing, pp. 43–55.

Zaidi, Syed Sahil Abbas; Ansari, Mohammad Samar; Aslam, Asra; Kanwal, Nadia; Asghar, Mamoona; Lee, Brian (2021): A Survey of Modern Deep Learning based Object Detection Models. https://arxiv.org/abs/2104.11892v2. Accessed March 29, 2021. *(Preprint submitted to IET Computer Vision)*

Zou, Zhengxia; Shi, Zhenwei; Guo, Yuhong; Ye, Jieping (2019): Object Detection in 20 Years: A Survey. http://arxiv.org/pdf/1905.05055v2. Accessed January 11, 2021.