UiA University of Agder

Computer Vision based Auxiliary System for Computer Assembly: System Design and Implementation

ERIK BØTUN & HENRIK BUGGE HALVORSEN

SUPERVISOR
Associate Professor Lei Jiao

University of Agder, 2019
Faculty of Engineering and Science
Department of ICT

UiA
University of Agder
Master's thesis

Faculty of Engineering and Science
Department of ICT

Abstract

This thesis proposes a solution that employs AI in assisting a human with little or no technical background for computer assembly in real-time without any form of other assistance. To achieve this goal, a state-of-the-art object detection, namely Lighthead R-CNN is adopted as foundation. Alterations and modifications of the algorithm are carried out to achieve the optimal trade-off between accuracy loss and speed gain. In more details, it is expected to reduce the complexity of the algorithm in order to make the solution applicable in a computer with limited computational capacity. Numerical results show that the proposed solution is 5.25% lower in terms of accuracy but almost 8 times faster compared with the competitors. In addition to the objective comparisons, we did subjective testing by inviting non-technical people to assemble different PCs with assistance of the proposed system. The results of the testing show a successful rate of 95.99%. It is worth mentioning that the unsuccessful cases are mainly due to the ignorances of the guide or the detector by human beings, rather than error output of the AI based solution.

Preface

This thesis is the result of the course IKT-590 Mater's thesis at the Department of Information Communication Technology, Faculty of Engineering and Science, University of Agder (UiA) in Grimstad, Norway.

We would like to express our thanks and gratitude to supervisor Associate Professor Lei Jiao from UiA for his supervision and guidance throughout the project. He has given valuable assistance and feedback in regards to the thesis content and research work. We would also like to thank him for his patience and trust in us when we had a sudden change of heart in thesis topic.

Table of Contents

List of Abbreviation

AR Augmented Reality. 3

CNN Convolutional Neural Networks. 11

CPU Central Processing Unit. 3

FPN Feature Pyramid Network. 11

GPU Graphics Processing Unit. 3

ITS Intelligent Tutoring Systems. 14

PSU Power Supply Unit. 3

RAM Random Access Memory. 3

ROI Region Of Interest. 17

List of Figures

Figures not cited are either taken by the group themselves or does not require admission as per stated by authors.

List of Tables

Chapter 1

Introduction

Desktop computers are very common tools as of today and can be found everywhere from offices to private owners. However, as desktop computers do not come pre-assembled as laptops do, they have to be assembled by someone with the required knowledge. Some vendors will build a computer as part of a package, but one will usually end up paying a little bit extra compared to doing the assembly oneself. A fee might not be directly attached, but buying different parts from different stores to achieve the best total price is not an option unless one can assemble a computer. Thus, one might turn to friends, family or even vendors to assembly a computer. However, the assembly of a computer is not a complex task, but might seem overwhelming to newcomers when met with the different components required. Even with numerous guides and how-to's online, if one does not possess a somewhat understanding of computers, it can be easy to turn to other solutions, which people earn money on. Thus, there is place for a solution that can guide and assist a complete newcomer in computer assembly as if a person were guiding them.

There already exists numerous guides on how to do assemble computers online. However, many tend to still let others do the work. There could be several reasons to the issue; even with a multitude of video guides on how to assemble, one will still have to decide oneself what goes where, but from what one understood from the video and thus uncertainty can arise. Another reason could be the same as of changing tires; Simple, yet many people can not be bothered to learn or do it themselves.

By introducing a new method, a guiding computer vision system, it can be interesting to see if people who usually are unable to assemble a computer are able to. If so, it can be a substitute for vendors offering assembly, as well as allowing purchase of components from different stores if one does not know how to build a computer.

Another aspect of the task is the idea of AI reproduction. Computers creating themselves might seem far fetched, although the project can be seen as a stepping stone. If AI is ever to reproduce, it has to know how to assemble itself and what goes where. While there are numerous other issues related to computer reproduction, we believe solving the problem at hand is one small part of it.

While the assembly of a computer might not be hard, creating a system that is able to closely guide a person to can be a challenge. Such a system should be able to tell the name and the feature of the component, where it goes on the motherboard and how it is connected. To be able to tell what a component is to someone with little to no knowledge, visuals has to be used. This approach brings various challenges with it. There are many different components in a computer when everything has to be explained. Furthermore, there are variations within each of these, such as DDR 3 and DDR 4, which means compatibility is also a factor. CPUs are not universal to all motherboards, and depending on visuals alone to understand compatibility might not be optimal. It has to be thorough, as every little stage where a user can become uncertain could result in failure.

The system should be live filming, as it would be obnoxious to take singular pictures for every stage along the process, as well as it would be more beneficial for a user to be actively able to see the area of interest. The system must be light as well as precise if it is to run in real-time and be functional.

Additionally, the usage of a state-of-the-art algorithm which in theory is excellent for the task proposed is planned [1]. However, while it does exist a paper on it, there is very little explanation on how to use it other than a code example proposed by the author, which is tailored to their specific case. Some have reproduced it, albeit the amount of information is scarce compared to other object detection implementations. There exist little to no actual implementation in an application. Thus, it is interesting to see how well it performs in real applications as well, rather than just their claimed results.

While there does exist similar solutions and guides on how to build a computer, there are no methods that use AI to understand the individual external components (CPU, GPU, cables, etc.) and internal components (CPU socket, power sockets, etc.), mapping them to each other. The most closely approach is a tutoring system combining augmented reality together with a computer tutoring module. However, this requires an AR setup and an external computer [2]. This covers the area of teaching computer assembly rather than the assisting and guidance of building ones own computer, which could be highly personalized. The main approach is rather to assist in the assembly of a computer, able to adapt to components present. What one learns from such an approach is however an indirect effect of this.

There are several important elements to solving such an issue; every component needs to be properly detected. A false detection (i.e. detecting RAM as a GPU) will result in a confused user. Detecting existing slots on a motherboard is also needed, as these must be mapped to the existing components the user possess. At this stage, compatibility is addressed to see if the components will actually fit onto the motherboard. Connectors and cables from the PSU, as well as inputs to the external components are also checked for, as these are steps along the assembly process that could confuse a user. Ultimately, the assisting system should make the assembly of a computer feel like LEGO for a user. For such a system to properly function, it must be light and accurate, keeping extra setup to a minimum.

There are some limitations to the system. It does not take computer cases into consideration, but rather the assembly of parts onto the motherboard itself. Connectors from a computer case onto a motherboard are indeed a thing but usually includes functions such as stock fans, external power-button and LED lights. It is not required for a computer to function and is thus excluded. Lastly, for compatibility to be possible, there has to be a limitation of motherboards supported. Thus, it will function as a proof and demo that it is indeed possible. Extending the model to detect additional motherboards is however possible. Lastly, the purpose of the project is to see if AI is capable of assisting a human in assembling the computer itself, not to connect external devices such as a monitor, mouse or keyboard.

1.1 Motivation

The main motivation for creating the solution is that nothing solves the issue as of yet to the best of our knowledge. While there exists similar approaches using AR, their goal is different and more importantly require additional hardware and setup. It is a learning environment more than an assistant. Thus, it is very interesting to see how well the proposed system performs with real people that have little to no previous knowledge of the inside of a computer.

Another reason is to be part of something that can introduce and help newcomers to the scene of computers. It can be overwhelming on where to start when building ones first computer, and while watching videos online on how to's exist, it is not the same as having a person guiding one through the process. This goes for young and old alike looking to enter the field. It can be a good starter tool.

Lastly, to introduce an additional implementation of Lighthead R-CNN to the scene, as there is scarcity of existing implementations as of now, especially in actual applications. Reasons to this could be caused by difficulties of implementing the Lighthead R-CNN algorithm from paper.

1.2 Thesis definition

This section describes the goals of this thesis, hypotheses on the issue and the fields of research it touches.

1.2.1 Thesis Goals

Goal 1: Create a system that can assist a person with low technical skills in assembling a computer through visual guidance.

Goal 2: Make the system able to detect compatibility issues.

Goal 3: Make the algorithm a hybrid between speed and accuracy for the best trade-off while still being able to complete the task.

1.2.2 Hypotheses

Hypothesis 1: It is possible to reliably detect components and sockets on a motherboard using object detection to assist in computer assembly in real-time.

Hypothesis 2: It is possible to develop a system that can assist a person with lesser technical understanding in assembling a computer through visual guidance.

Hypothesis 3: It is possible, to some extent, to replace vendors offering similar services with such a solution.

1.2.3 Field of research

This thesis touches several interesting fields such as computer vision, artificial intelligence and cognitive learning.

1.3 Thesis Objective

The objective of this thesis is to see if it is possible to create a solution which can help newcomers to the field to assemble computers through visual real-time guidance, even with little to no knowledge of computers. It will detect individual components, map them to the corresponding socket on a motherboard while also checking for compatibility.

1.4 Contributions

- A system that is able to assist humans with little to no previous knowledge in the field of computers to assemble a computer through visual real-time guidance.
- Experiments to show that such an approach works and can potentially be extended to replace existing solutions.

- An implementation of Lighthead R-CNN in an application, which exists next to none before.
- An implementation of Lighthead R-CNN which is able to do real-time tracking.
- Additions to the field of annotations.

1.5 Thesis outline

The report has the following outline:

- Chapter 2 talks about theoretical background of the research area, outlines the previous work in the field and the state-of-the-art.
- Chapter 3 explains the proposed solution.
- Chapter 4 details the experiments and results.
- Chapter 5 concludes the thesis and summarizes the future work.

Chapter 2

Background Theory and State-of-the-Art

Computer vision and object detection is a field with considerable amount of research and differing approaches. Where some optimizes for accuracy, others go for speed. While there are improvements overall to the area, it is usually required to sacrifice one for the other. Thus, there is a multitude of methods approaching the field differently. The best choice is often dependant on what the scenario is. In this section, the different work relevant to object detection is explained to understand what this thesis is based off and knowledge required for later chapters. There are numerous variants but only the theory of a handful are explained as they often take ideas from each other, making many of the approaches somewhat similar. It should be sufficient to gain an understanding of how the field of object detection functions.

2.1 Object Detection

Object detection, much as the name states, is a technique to enable a computer to detect specific objects in an image. They take an image as input and, based on the network used and design, the detected object of interest [3]. Most of these techniques share a similar start path but differ as they go on. First and foremost, they are deep neural networks and are based off the original idea of how to understand an image. How they differ is often through design and implementation choices rather than the model itself. This section describes the most common features object detectors share.

2.1.1 Detection Type

When an object is detected, it is important how it is represented. The main method used is through bounding boxes, which are rectangles surrounding the object of interest [3]. However, if one wants to get the specific silhouette of an object, this is not sufficient. Thus, semantic segmentation exists which applies a detection specifically on the object with a polynomial structure.

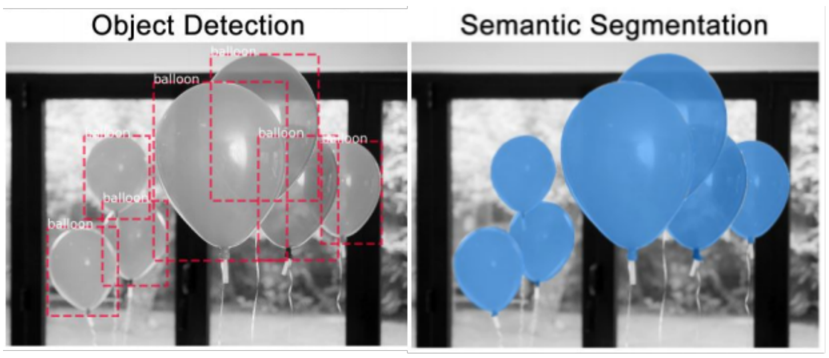


Figure 2.1: Difference between object detection and semantic segmentation [4].

2.1.2 Loss Function

The loss function is important as it is how the algorithms weight these boxes differently. For example, the difference of a few pixels on a small box can be made significantly more impactful than if the bounding box is large through loss functions. To further enhance this, weights can be applied to the different loss functions for better result, depending on what is needed [3]. While they all have these functions, this is a part that might vary greatly and create distinction between the approaches.

2.1.3 Feature Extraction

When understanding an image, a lot of raw data is processed. If the entire network had to process all the data from an image, it would be very slow. Thus, feature extractors are used to find the most impactful features in an image by combining many variables into features, for example a specific shape [5]. As such, speed is achieved at the loss of less important details. Some popular feature extractors are VGG, Resnet and MobileNet [3]. They are often referred to as "backbones", largely representing the rest of the network in terms of accuracy and speed. What to use depends on the scenario in question, as some have a focus on speed whereas other prioritize accuracy. This goes for the detector as well.

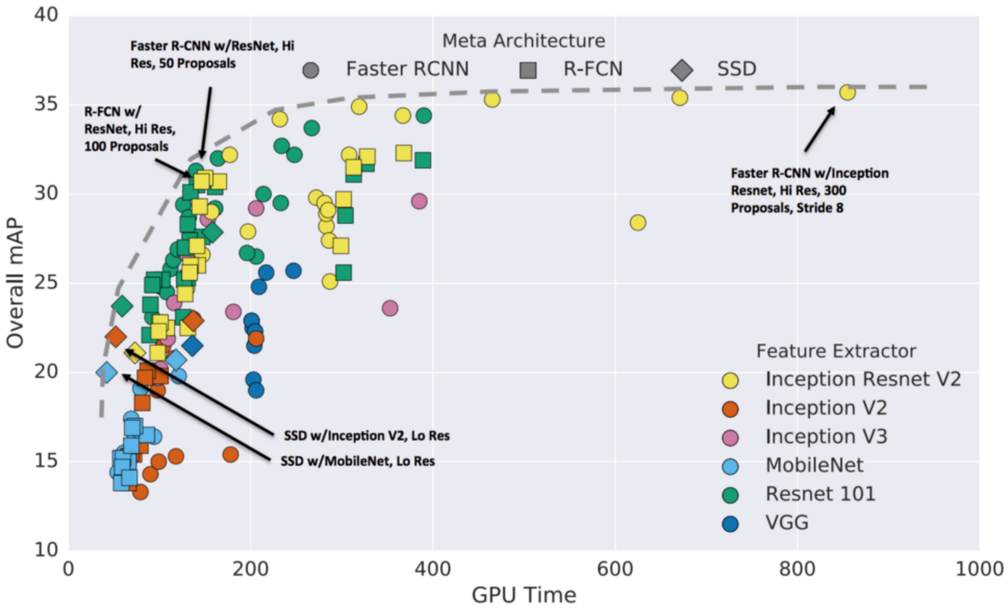


Figure 2.2: Accuracy vs time, detector model and feature extractor [4].

2.1.4 Annotation

When training an AI in object detection, a specific type of data is required. Parts of it are done through annotation, which in essence shows the AI a "ground truth", what to look for in an image. This is done through silhouettes of an object or bounding boxes. The process of annotation is a manual, time-consuming process. This means having multiple classes to detect and large datasets will drastically increase the time spent on annotating. There are large pre-annotated datasets such as ImageNet or COCO, which include common objects [6] [7], albeit it does not include computer components. While it may not include a specific object one wants to detect, it does make for great starter weights (i.e. understanding electronics, but not specific parts). To create annotations a tool is required [8], which there are many options. An image together with its respective annotation file creates the foundation of the training and validation data.

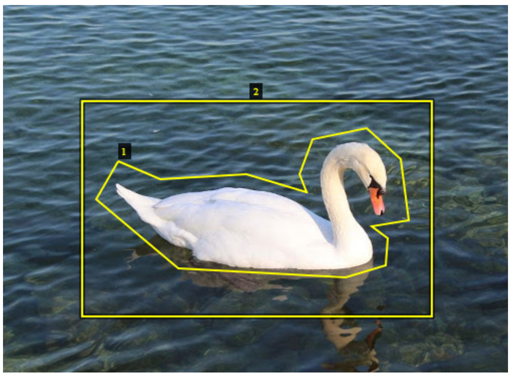


Figure 2.3: Annotation of a swan using polygons and a box [8].

2.2 Two-Stage Object Detector

While there are many object detection methods, they can be divided into two; two-stage detectors and one-stage detectors. The two-stage detectors mainly consist of the R-CNN family, a region-based approach to object detection. The idea is to first propose a set of regions that have interest through a “regional proposal network”. The next stage is to only process these proposed regions for classification. This methods allows for detection of small details and usually result in a higher accuracy [9]. However, since there can be a high amount of regions proposed, they are usually slower than one-stage detectors (albeit only one of the reasons as to why).

2.2.1 Region Proposals Family

The region proposal family is what the two-stage object detectors mainly consists of. Some popular ones are R-CNN, Fast R-CNN, Faster R-CNN and Mask R-CNN [10]. They are conceptually alike, but each iteration have improvements or modifications to the previous [11].

Similar to other methods, they use a feature extraction. However, a network called Feature Pyramid Network is used as well [12]. It is handy to use when objects or details of interest are located in very different size and location in an image, as the algorithm might try to classify the object based on size, not its features.

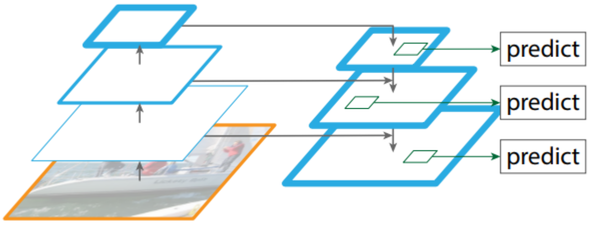


Figure 2.4: Latest design of a feature pyramid, the FPN [12].

The bottom-up part of FPN is a regular CNN for feature extraction as mentioned earlier. The further up, resolution decreases. Strong features will persist, but lesser will disappear at the top layers. The network will reconstruct images from these layers, creating features based off the spatial resolution decrease. However, up and down-sampling means objects will not be precise in terms of location. Thus, connections between the reconstructed layers and the corresponding feature map are added to help predicting a location better [12].The idea is to take an image, alter the size and create features at different scales.

Next is the algorithm-defining regional proposal network [13]. Its purpose is to check if a section in an image may contain interesting objects by using the feature map produced from the FPN as input. A box is slided over the input feature map and creating “anchors” [10] [11]. These anchors are at the center of these boxes, through altering the size ratio of these boxes many different segments of the image are covered and checked for interesting features. For each anchor, several other anchor boxes are created as well to cover area.

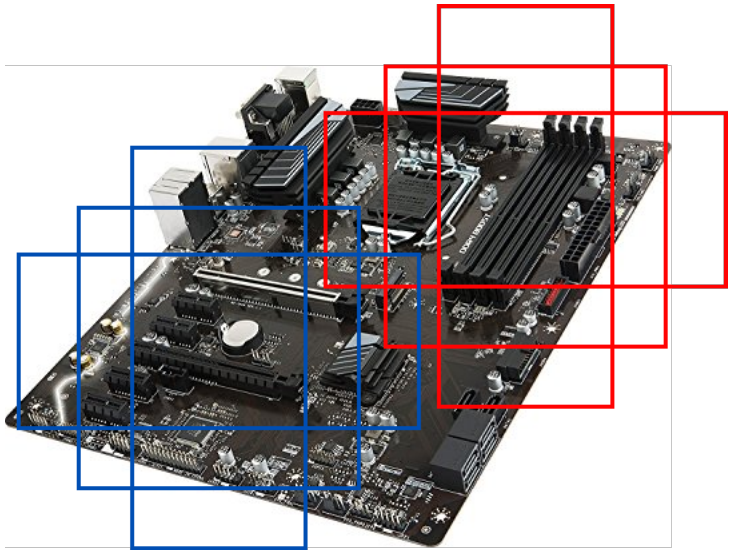


Figure 2.5: Anchors placed on an image and its respective anchor boxes

Scanning anchor boxes can be somewhat fast, as it uses the feature map from the FPN to prevent redundant calculations. However, even if increasing the amount of anchors created for each image might increase accuracy, it will decrease the inference speed.

2.3 One stage Object Detectors

Instead of using one network to propose regions and second one to fine-tune and predict from these proposals, one-stage predictors use a fixed number of predictions on a grid. There are no intermediate tasks that must be performed before it can produce an output [14] [9]. This is faster and usually leads to a simpler architecture, albeit it can somewhat reduce the accuracy. Much like two-stage detectors, it starts off feeding an image to a feature extractor to obtain the features of an image.

2.3.1 You Only Look Once

As seen from earlier, other methods use the region proposals approach. A popular series of networks called "YOLO" will only look at an image once [15]. The network splits an image into a grid, 13 × 13 cells, each which will generate bounding boxes B encapsulating an object. These are just predictions, thus a confidence score that actually tells how certain the box contains an object is used. If said box has a confidence high enough the area contains an object, it may predict a probability for the object belonging to classes specified. In total, C × C × B boxes are generated, far less than of two-stage detectors [9] [14]. Thus, it is able to do inference at a high speed.

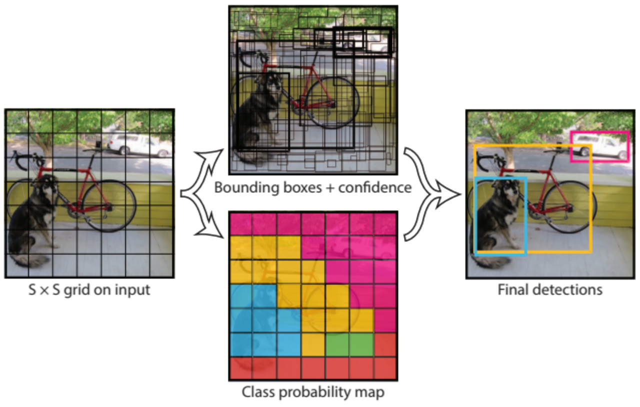


Figure 2.6: High-level explanation of how YOLO works [16].

YOLO uses a custom backbone developed by the same person developing the YOLO-series, called "darknet". Essentially, it is a small neural network framework used for feature extraction just as ResNet or VGG.

2.3.2 Single Shot MultiBox Detector

Normally, CNN networks will shrink the feature map as it goes through deeper layers. The deep layers covers larger receptive fields, whereas the shallow layers cover smaller fields. SSD's uses this and lets shallow layers predict smaller objects and deeper layers to find big objects [9]. However, as these shallow layers might not generate enough high-level features, SSD performs worse at predicting small objects rather than large ones [17].

SSD uses multiple feature maps from an FPN [12], but discards bottom layers as they are too high in resolution. The accuracy gained is lackluster to justify the loss in speed, as this is significant (region proposals family networks use the entire pyramid, thus loses speed). Hence, SSD only uses upper layers of an FPN for detection [17].

Both YOLO and SSD are one-stage detectors, although there are some clear distinctions. SSD does not predict a value for the probability of an object and then the class label, but rather directly predicts a class being present in the given area scanned. This is different from YOLO-networks, which first looks for the chance of an object being present, thereafter predicting the object's class [14].

2.4 Related work

The most closely related work to this project is a system consisting of augmented reality (AR) combined with an ITS, namely Motherboard Assembly Tutor (MAT) [2]. The goal of the system is to assist with training for manual assembly tasks, somewhat covering the goals of this project. However, its limitations are clear: An AR setup is required as well as the ITS system. This means a person capable of setting up said system is more than likely able to assemble a computer as well. Furthermore, it does not detect individual components through object detection, but rather by using "multimarkers" seen below. It serves multiple purposes such as relativity of one object to another as well as drawing 3D parts.

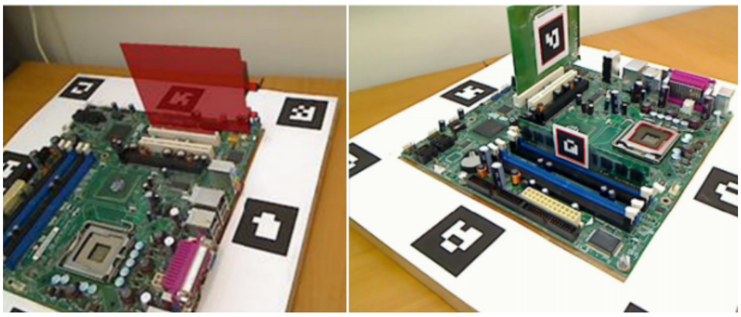


Figure 2.7: Multimarkers can be seen as the black and white squares [2].

While it does a good job in providing a learning environment, it is not capable of providing assistance to a newcomer without additional human support or previous setup, which this projects aims to solve.

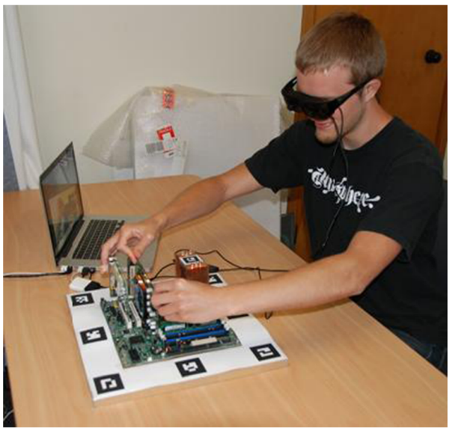


Figure 2.8: Setup of the MAT environment [2].

2.5 State-of-the-art

This section describes the most recent prominent models from the different approaches to object detection. More could be added, but these are seen as the most relevant to this thesis.

2.5.1 Mask R-CNN

Mask R-CNN, a two-stage detector, is one of the latest additions to the region proposals family. It is heavily based off Faster R-CNN, but with some extensions and modifications. It adds a branch called masks, essentially adding masks upon the already existing bounding boxes. Thus, it is capable of generating predictions inside these bounding boxes, resulting in silhouettes of the object. Ultimately, it is capable of reaching very high-quality segmentation of objects, but at the cost of speed [10]. Thus, mask R-CNN usually only excel when speed or real-time detection is not a criteria.

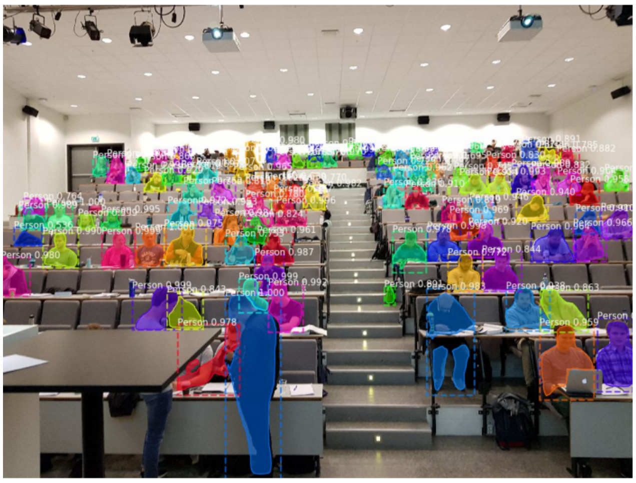
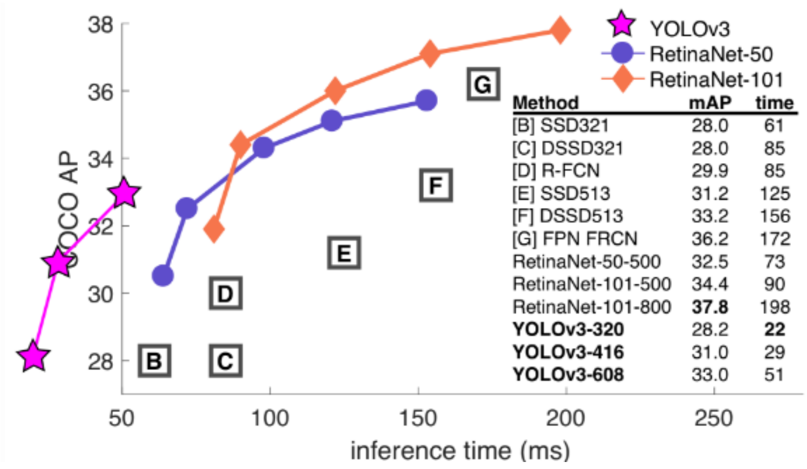


Figure 2.9: A prediction of a crowd with bounding boxes and masks applied.

2.5.2 YOLOv3

YOLOv3 is the latest version in the YOLO-series. While being more accurate, it has lost some of its speed [18]. However, this means it is able to compete with algorithms performing better accuracy-wise, but at a much higher speed than these [15].



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 2.10: Comparison of YOLOv3 and other popular detection models [18] [19].

The backbone and underlying architecture of the model has been changed. Previous iterations had issues with detecting smaller objects caused by the size of the small network and lack of features. Residual blocks have been added and it now uses Darknet + ResNet for a total of 53 layers, naming it “Darknet-53”. This can be doubled for accuracy, albeit this negatively affects the speed.

The version has dropped its softmax function for class predictions; this is because a softmax function assumes classes are mutually exclusive [20]. However, the assumption can be false for certain datasets, since a class banana can be a class food. Furthermore, it adapts the FPN from the RPN-family to detect objects of lesser size. Lastly, it predicts at three scales rather than one, downsampling the input image by 32, 16 and 8 [18] [9]. Hence, a lot more bounding boxes are generated and the accuracy of detecting smaller objects is higher. However, these additions to the network does slow it down compared to previous versions.

2.5.3 Lighthead R-CNN

While algorithms from both one and two-stage are good at certain things, they all have their drawbacks usually related to being one or two-stage. This is where Lightead R-CNN comes in, being a two-stage detector. It uses the concepts and methods described earlier from two-stage, but tackles the issues that two-stage detectors usually have, a lesser speed. The headers of these algorithms are often the issue, as they have large feature maps that have a cascading effect further down the network. Large feature maps means more work for the [13], actual regions of interest, ROI-pooling and so forth. Lighthead alters the heavy header by using a thinner feature map. Replacing the backbone with tinier networks such as Xception rather than ResNet50/101, allows it to become the fastest performing algorithm of both one and two-stage detectors [1]. Using a heavier backbone makes it slower, but in return higher accuracy. Thus, Lighthead R-CNN is a flexible model that can be fitted into many scenarios.

Specifically there are primarily two parts making the network fast and accurate, being RoI warping and the small R-CNN subnet. Other algorithms tends to use heavy headers, whereas Lighthead R-CNN utilizes a simple and cheap fully-connected subnet. The process starts off with the network proposing the thin feature map. This, in contrast to for example R-FCN, is done by large separable convolution. The next part is RoI-warping on the thin feature map. In essence, this warps a certain RoI on the feature map to a fixed dimension, which changes the shape of the feature map [1]. According to the authors, this results in an increase in both accuracy and speed [21]. Furthermore, RoI-pooling (for each RoI taken as input, a section on the feature map corresponding to the specific RoI is resized to a fixed size) is applied to reduce the overhead proposed to the R-CNN subnet, which significantly improves performance.

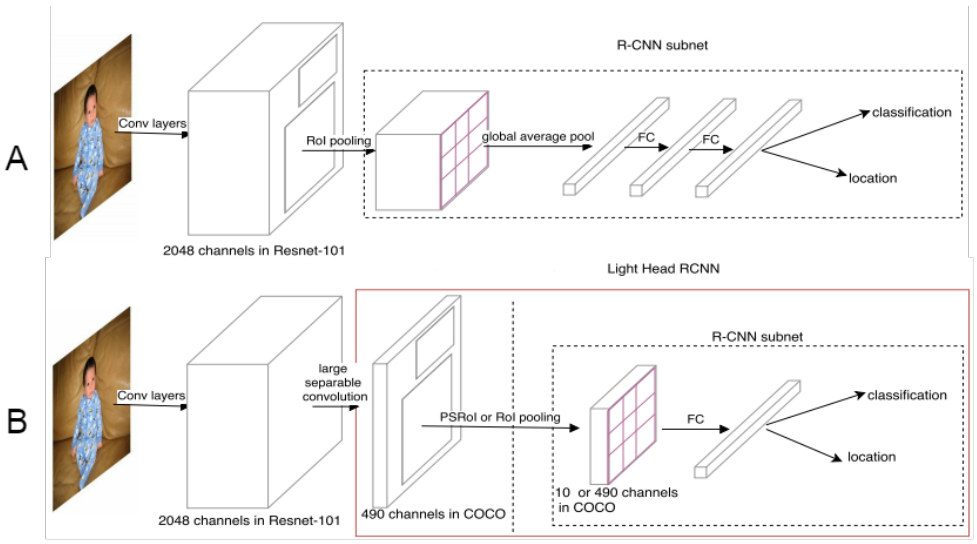


Figure 2.11: A is a Faster-RCNN structure, B is lighthead-RCNN's structure [1].

Chapter 3

AI based Auxiliary System

This chapter aims to describe the solution proposed to solve the issue at hand, as well as why these decisions were made.

3.1 General Approach

As stated earlier, the problem is to create a system that is able to visually guide a newcomer to assemble a computer in real-time without human support. Hence, object detection is a good starting point. A person does not need to know what the individual parts are or where they are supposed to go, but simply scan the area and the specific part required will be automatically detected. From there, they will be told where to place it through real-time visuals. The idea is to use a simple camera such as those in mobile phones.

If the system is to feel real-time, it has to be capable of handling around 7-10 frames per second. Therefore, the network has to be lightweight. Furthermore, when deciding on what to use, the results proving the performance of different models can be somewhat misguiding to our scenario. One cannot expect a user to be in possession of multiple powerful GPUs. However, it needs to be accurate as well and the trade-off between the two is important to be aware of. One approach is to solve the issue by connecting the phone to an external computer and let the external computer do the predictions.

Naturally, this will result in higher accuracy as it allows for a heavier network as more computational power is available. However, this means an external computer for computation is required. This is the approach used in the project.

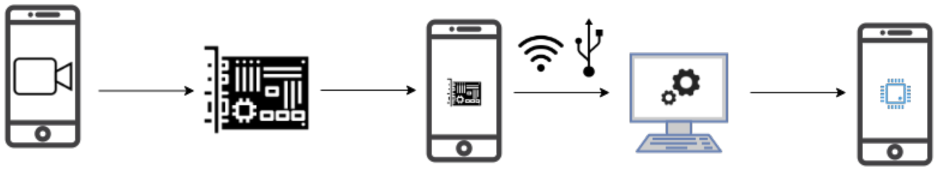


Figure 3.1: Approach A, using an external computer.

In a real application instead of using an external computer present at a users location, 5G can be utilized. From a mobile phone the data can be transferred to an edge computer in a base station of a 5G network, offloading the workload the external computer located at a users location has to perform. As such, the computational power required is outsourced to the edge computer and a user will only require a cellphone to use the application while avoiding the issue of delay if using a regular cloud solution were introduced.

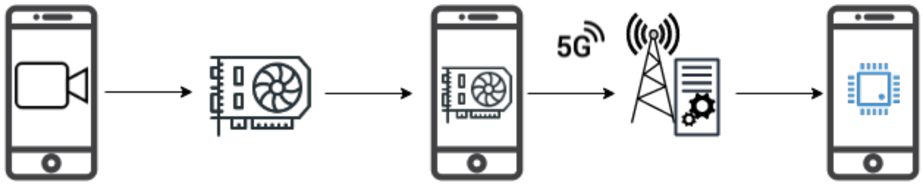


Figure 3.2: Approach A, using a 5G edge computing network solution.

Another approach is to make the network so small it is capable of running on mobile devices. An obvious downside of this is the drop in accuracy, but gain the ability to not be dependant on any external support other than the download of an application. However, such an approach might lose too much accuracy to the point where it cannot provide reliable support.

3.1.1 Step-by-step Guide

As the system is supposed to be an assistance to newcomers, it is important it does not feel overwhelming to use. Thus, proposing every class simulta-

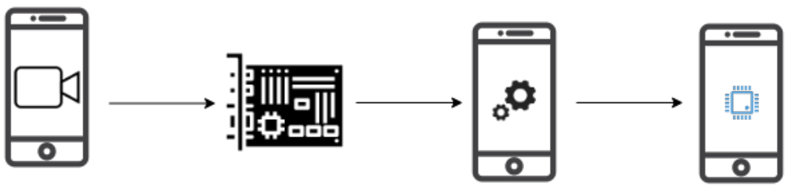


Figure 3.3: Approach B, only using a phone.

neously is a bad idea and would look like a mess of detections at the screen at once. Even with color codes for classes, it would be too hard to see and ultimately result in confusion. Therefore, a guide following a linear pattern is proposed. Its aim is to solve each step individually to minimize the potential of failure in assembly.

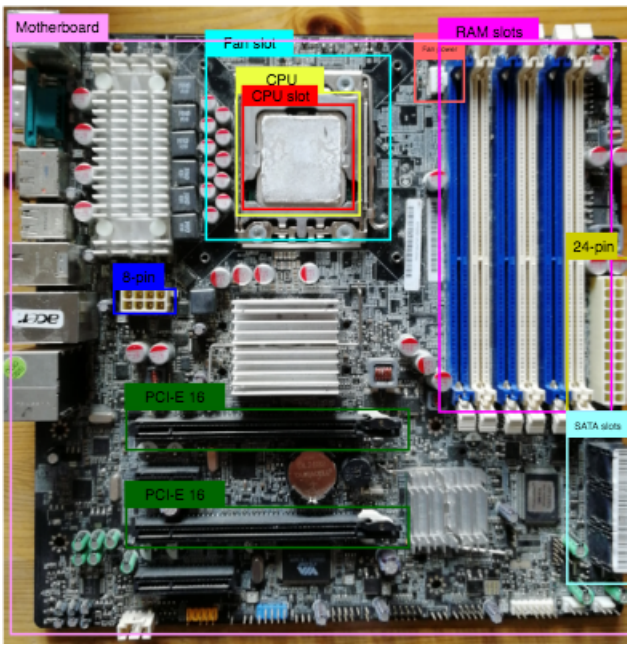


Figure 3.4: Mass detection, with even more instances confusion can arise.

This is done through deactivation of every other class other than the one in question, i.e. at the CPU stage only a detector of the CPU is active. From there, the corresponding CPU socket will be activated, deactivating the detection of the CPU itself.

It is assumed the parts are laying around visible to the camera.

- The initial stage looks for the specific motherboard. This allows to check for which type of RAM is supported, CPU socket type and supported CPUs.
- Next, scanning of external parts is done, looking for RAM. The system will detect the RAM and which RAM-type (DD3 or DDR4). Since it knows which type of RAM the motherboard supports, it can tell compatibility-issues at this point.
- Next is scanning for a RAM-slot, mapping the RAM and RAM-slot together.
- User is told to check the CPU package box, then look specific information displayed on the box (what generation and socket-type) and see if this matches the proposed information obtained through the scanning of the motherboard.
- CPU is detected as well, in case a user does not possess the box but has the CPU (albeit this will not allow for compatibility check).
- Detecting CPU-socket.
- Detecting CPU fan.
- Detecting Fan Slot.
- Detecting Fan Power slot.
- Detecting Fan Power cable.
- Detecting PSU.
- Detecting 8 and 6 pin power connectors (GPU).
- Detecting GPU.
- Detecting PCI-E 16.
- Detecting 8 and 6 pin power slots (GPU).
- Detecting 24 pin power connector.
- Detecting 24 pin power slot.

- Detecting 8 pin power connector (CPU).
- Detecting 8 pin power slot (CPU).
- Detecting S-ATA cables.
- Detecting S-ATA slot on motherboard.
- Detecting SSD/HDD.
- Detecting SSD/HDD S-ATA input.
- Detecting SSD/HDD power cable.
- Detecting SSD/HDD power input.

There are explanatory steps between some of the steps where it is required, however this is roughly the steps taken by the system to guide a user through assembly. It is assumed the user is able to fit a component and will ask for the next step when finished with the current task. Further, the task is to assemble a computer, not connect to a monitor, power outlet and other devices which is why those potential steps are excluded.

3.1.2 Compatibility

The compatibility primarily boils down to the CPU and RAM, being several ways on how to approach this. For the CPU, it is possible to check for the specific socket type present. However, the level of detail in difference between socket types can be very minuscule. Furthermore, detecting a specific socket type is not enough; different motherboards only supports certain types of CPU, regardless if the socket in question fits. This is somewhat different with RAM, where it usually only supports the RAM slot type present on the motherboard. Hence, detecting the specific slot type present would solve this issue. However, the problem of supported CPU types still persist.

A solution to cover both above issues exist; by looking at the motherboard itself instead. Through this, one can use a lookup on what type of CPU (i.e AMD or Intel), socket type and generation support. Further, supported RAM versions can be checked for through lookup as well. There are however downsides to such an approach too. A user will have to check the box the CPU came in manually, looking for the information provided by the lookup check. For example, even though the system detects the specific motherboard and provides the user with its compatible versions, a user will have to manually crosscheck if the information provided matches with what they have. However, this should be a minor issue as this information tends to be very visible, as well as the system telling the user what to look for. Additionally, this means only motherboards told to detect for are supported (albeit this can quite easily be extended) and a database for lookup is required.

3.1.3 Database lookup

A MySQL (XAMPP) solution is used for the database. The system detects which motherboard is present and queries the database for its compatibles. It returns the socket type, supported CPUs and the RAM supported respective to where it is in the guiding process. For the purpose of demonstration, only three motherboards are used.

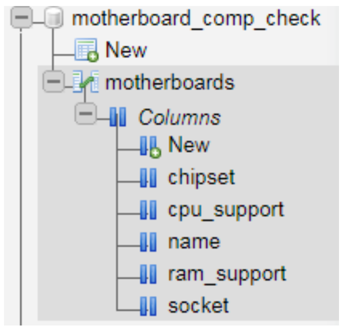

motherboard_comp_check
New
motherboards
Columns
New
chipset
cpu_support
name
ram_support
socket

Figure 3.5: Structure of the compatibility-check database.

3.1.4 Prediction and Tracking

To track objects, a predictor for the model is built so it can receive video input. The input given is from an external live camera and each frame received is run through the model. For the prediction to go smoothly it relies on the network being lightweight. Frames per second from the external live camera is lowered to fit the inference time to minimize the delay. If too many frames arrive at the predictor, it will discard these frames. This is done as there is little interest in detecting an object seconds ago, and every frame is not needed to understand the object of interest.

The predictor is built to properly fit the task at hand. The weights are loaded into the model at the start of the prediction, albeit only the classes described in the specific step are activated. When a class is activated and found, its location and name is applied to the frame and displayed. Every class has its own threshold. This means the predictor needs to adjust the requirements of a class to be detected. This is implemented due to a difference in detection between classes, where some are detected easier than others. Further, it functions to reduce the chance of false positives where some classes may look similar at specific angles. Information regarding the steps are located on the top of the screen when the user is scanning for objects. Information provided changes dynamically in the guide, as information is based off the motherboard scanned in the start. The respective information and compatibility is grabbed from an external database according to the motherboard detected.

3.2 Data Set

A total of 29 classes are used (excluding the background) with a total of 1396 instances, split into external and internal. The internal section covers the classes that cannot be removed and are found on the motherboard itself or components. It also includes the motherboard. External classes are the loose components placed by the user. The classes will be mapped to their corresponding counterpart. See appendix B for a complete overview of the classes.

3.2.1 Data Gathering

There are no image database that specifically have computer components, especially not something so specific as the sockets or pins on a motherboard. However, some can be found on more general free image databases such as shutterstock or pixabay. Other good sources of data is the websites selling these types of hardware, as they usually have many different images of the hardware as well as different angles of it. Lastly and the main source of data, the group has taken pictures themselves to get specific instances of

Part	Instances
ASUS Z97-AR (motherboard)	17
MSI Z87-G45 Gaming (motherboard)	21
Gateway TBGM-01	23
PCI-express x16	171
CPU socket	59
6-pin power connector slot	48
8-pin power connector slot GPU	28
8-pin power connector slot CPU	67
24-pin power connector slot	70
S-ATA slot	129
CPU fan slot	40
Fan power slot	63
SSD/HDD power slot	56
RAM slots	75

Table 3.1: Internal parts

Part	Instances
SSD	34
HDD	19
DDR3 RAM	105
DDR4 RAM	19
SSD/HDD power cable	23
GPU	34
CPU	30
CPU fan power connector	45
CPU fan	29
PSU	31
6-pin power connector	34
8-pin power connector GPU	23
8-pin power connector CPU	10
24-pin power connector	46
S-ATA cable	36

Table 3.2: External parts

each class from varying angles. This helps when trying to detect objects real-time, as the pictures are taken in a similar scenario as to how a user will use a camera to detect an object.

Having images alone is not enough; annotation is needed which is described in chapter 2. This has been done manually by the group on the images gathered through above mentioned method. The tool used in this project to create these annotations is VGG Image annotator [8]. There is a total of 1396 annotations split over 29 classes.

These images, together with their respective annotations, will be given as an addition to datasets such as COCO [7] and ImageNet [6], as the data used in this thesis is lackluster elsewhere.

3.2.2 Annotation Format

Originally, the annotation files created through the VGG tool comes in a JSON format. Moreover, it contains a lot of unnecessary information not used in this project. Hence, a script to strip these annotation of information besides what is an absolute necessity has been created. By doing so, the file size of individual annotation file decreases, which in some cases adds up fast if large amount of pictures are used for training and validation. It also speeds up the process of loading said files. Lastly, the files are converted from JSON files to ODGT (Object Detection Ground Truth) files, which in essence are text files.

3.3 Lighthead R-CNN

There are several issues to be aware of when designing such a solution and choosing an algorithm. Two-stage object detectors are primarily focused on accuracy. While this is good for detecting larger objects as well as detailed one, they have the downside of being slow. This is a deal breaker when making a system which must be light and real-time. The one-stage detectors are usually fast, but lack the ability to properly detect object smaller detailed objects. This is also not ideal, as many of the computer components, especially the ones considered internal, are very small with some distinguishing features.

Fortunately, there are some algorithms that are hybrid in terms of accuracy versus speed, in this case Lighthead R-CNN. It is, as described in chapter 2, a state-of-the-art two-stage object detector able to be both fast and accurate, whilst also being flexible to suit the preferred scenario better. Mainly, the network achieves this through lighter headers than other two-stage algorithms as larger ones incur heavy calculations deeper into a model. However, having a good algorithm is only a starting point. If it is to run close to real-time on realistic hardware, further stripping and modification of the network is required as direct implementation will result in a slow inference speed [22] [23].

3.3.1 Backbone

Backbones, described in chapter 2, significantly impacts the accuracy and speed of a network. Lighthead R-CNN is usually based off ResNet101 for its high accuracy results and Xception-like networks for its speed. However, these are tested on computers with powerful GPU's. To make it even lighter and faster, a tinier backbone can be used but at the cost of accuracy, meaning there has to be a middle ground for it to still be viable in detecting objects with details. This does of course depend on the approach used (see figure 4.1 and 4.2), where approach A allows for a heavier network than B does.

As stated above, even though ResNet101 might seem to be an ideal solution for a backbone, it struggles to achieve a high FPS when proposed with detection of a live video feed. Furthermore, it requires significant computational power to do so as well. Include a large amount of classes and the size of the trained model increases drastically. Usually, such a backbone fits more to a more static type of object detection that does not depend too heavily on speed and is normally seen when reaching for high mAPs (see 4.1.1). To be able to detect in real-time with lesser computational power, as this projects aims to solve, a lighter backbone could be used. Some popular ones fitting to the scenario in question are MobilenetV2, ShufflenetV2 and Xception, each with their pros and cons [24] [25].

At the current state, it seems obvious to downgrade the backbone to something lighter as mentioned above. However, by looking at different frame rates, one can realize a high FPS is not required to effectively understand where an object is in real-time. The difference between 20, 10 and even 5

FPS is very noticeable, but of lesser importance when trying to understand the location of the object or what it actually is [26]. Hence, a real-time feel can still be achieved even at lower FPS, which in turn allows for a heavier network. This means the accuracy and detector is better, but at the cost of speed which is not needed anyway. Hence, a Resnet101 backbone is used instead of downgrading. To make up for the loss in speed caused by using larger networks, some other alterations are done to parts usually responsible for larger computational time. The amount of anchors created per image have been reduced. Boxes proposed potentially containing object of interest have been reduced per image, resulting in a lot less to predict and scan over. It lessens the ability to detect objects where details are important and where the detector is somewhat uncertain the area contains anything of interest. The amount of proposed regions is reduced as well, this combined with smaller images drastically increases the frame rate. Simply put, having to scan over fewer areas in a smaller area is much faster than having to scan many times in many areas over a large area.

3.3.2 Stripping the Model

Alternations are done in advance of training to reach the goal of a fast and accurate model. As the backbone does consist of many parameters, the optimal solution would be to reduce the amount of data stored. The variables of interest need to be adjusted in a correct manner to keep balance between the preferred accuracy and speed.

The initial algorithm creates large weights, resulting in a bad inference time when using a heavy backbone such as ResNet101. One solution to achieve better inference speed is a reduction of weight size and parameters produced. The second part is reducing potential scanned areas. Anchors, as explained from above, are boxes of varying aspect ratios shifted over an image. Each ratio has several scales, and each scale can have many bounding boxes, being a prediction to be refined. This goes for every anchor produced as well, creating exponentially larger weights if a large amount of anchors are used in combination with larger images. Hence, to produce smaller weights an aspect ratio of anchors are set to 1:2, 1:1 and 2:1. Further, in direct correlation with image size and the objects of interest, a scaling of 8^2, 16^2 and 32^2 is used. This means the maximum size an object can be detected at is 1024 pixels, while the minimum size is 64 pixels.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters
Xception	88 MB	0.790	0.945	22,910,480
VGG16	528 MB	0.713	0.901	138,357,544
VGG19	549 MB	0.713	0.900	143,667,240
ResNet50	98 MB	0.749	0.921	25,636,712
ResNet101	171 MB	0.764	0.928	44,707,176
ResNet152	232 MB	0.766	0.931	60,419,944
ResNet50V2	98 MB	0.760	0.930	25,613,800
ResNet101V2	171 MB	0.772	0.938	44,675,560
ResNet152V2	232 MB	0.780	0.942	60,380,648
ResNeXt50	96 MB	0.777	0.938	25,097,128
ResNeXt101	170 MB	0.787	0.943	44,315,560
InceptionV3	92 MB	0.779	0.937	23,851,784
InceptionResNetV2	215 MB	0.803	0.953	55,873,736
MobileNet	16 MB	0.704	0.895	4,253,864
MobileNetV2	14 MB	0.713	0.901	3,538,984
DenseNet121	33 MB	0.750	0.923	8,062,504
DenseNet169	57 MB	0.762	0.932	14,307,880
DenseNet201	80 MB	0.773	0.936	20,242,984
NASNetMobile	23 MB	0.744	0.919	5,326,716
NASNetLarge	343 MB	0.825	0.960	88,949,818

Table 3.3: Popular backbones, accuracy refers to tests performed on ImageNets validation dataset. Accuracy can differ on the test used and modification of the backbone (i.e. xception is lighter, but resnet101 can usually be made heavier and more powerful [27].

Each frame is allowed a maximum of 9 anchors and each anchor can produce no more than 350 bounding boxes. In theory, this means a maximum of 3150 bounding boxes can be produced per frame, each which has to be further refined by the RoI network. It loses the ability to detect objects in images of higher quality as the max size is 1024 pixels, albeit this is a non-issue in the task at hand as images are smaller than the maximum detection box. Increasing anchor scaling to include 64^2 and 128^2 allows for extreme detection, but at a high cost; Having a max size of 16384 pixels requires significant computational power, especially when considering the additional amount of bounding boxes per frame per anchor required to cover such an area.

The image size in training is a variable that has a large impact on the model regarding accuracy and speed. Dynamically, the shorter edge of an image is forced to 800 pixels and a longer edge is reduced to 1000 pixels. This specific change is done to minimize the loss of details in the images and gain a fair amount of speed. Using less pixels adds less data to the weights. If the image size is further reduced the speed would greatly increase, but it does sacrifice a large amount of accuracy which is not feasible in this setting.

The combination results in roughly 15% reduced weight size produced in training, together with anchors specific to the images used in the project and a significant less amount of data to be scanned in each frame when predicting, ultimately meaning a large gain in inference speed. As seen in section 4.1, the loss in accuracy is fairly small compared to the speed increase.

3.4 Preparing the model

In preparation to training, the model was changed from being based on the COCO dataset to be based on the motherboard dataset. All of the parts based on the COCO dataset of the model have either been fully removed or changed.

The ODGT (Object Detection Ground Truth) file consists of image name, bounding boxes, class names, image size, and the image path of each image. The model needs all these variables to start training. Each image is set be around a specific resolution to avoid large scale resizing when it is used in training. If the image is too small or too big the resizing of the image could lead to some degree of loss.

The number of classes used in the model are 29. The learning rate is set before training but changes throughout. The model consists of 30 epochs, and on epoch 25 the learning rate is multiplied with 0.1 and on epoch 30 the learning rate is multiplied by 0.01. The reason for this change in learning rate this late is due to avoiding big adjustments of the weights at the end of training. The full code is available on github. [1]

[1] `https://github.com/ebotun/lighthead_rcnn`

Chapter 4

Experiments and Results

In the report two approaches has been tested, first performance of the network. This goes in regards of the inference time, frames per second achieved and the accuracy of the detector.

The second part of testing is related to use of the application. The test aims to see how well the solution performs on assisting humans in computer assembly. As there are little amount of previous testing on the topic, test parameters included have been created by the group themselves to what seems logical.

4.1 Performance of the Detector

This section explains the experiments and results regarding the performance of the network. In the implementation proposed, the experiments are performed on real-time data. This means a camera is feeding the detector with a live video stream. The ability of the detector to detect the object of interest is what to be tested. However, such an approach makes it somewhat hard to compare to other approaches, which is explained in section 4.1.4.

Further, the detector is locked to a certain frame per second. The reason behind this is accuracy; By sacrificing speed and inference time, an increase in accuracy and better detection is achieved. However, they have to be at an equilibrium. Even though real-time and frames per second is different, a frame rate of 1 is hardly usable. Furthermore, it is unnecessary to have a frame rate of 20 [26]. At that point it is more a quality of life aspect than providing better support for the user. As such, the detector is optimized to the scenario of achieving 7-10 frames per second at an image size of 1000 × 800.

4.1.1 Explanation of Mean Average Precision

When evaluating different models and implementations, there is a different approach on how object detection is measured. Essentially there are two parts to measure:

- Decide if there is an object of interest present in the image or not.
- Decide where the object of interest is located.

Furthermore, a multitude of classes might be present. Imagine its task is to detect cats, dogs and pizzas, but there is a majority of cats and dogs. It could in theory be extremely bad at detecting pizzas, but since it is amazing at detecting cats and dogs a simple accuracy approach will show biased results. Hence, a confidence score is given to each box detecting an object, essentially saying “I am 60% sure there is a cat inside this box”.

Because of the issue above, an Average Precision (AP) is used. AP uses precision and recall from the model used. Keeping it short, precision in a detector measures the false positive rate (this means the ratio of true objects detected to the total objects the model has detected, the percentage of the predictions which are correct) [28] [29]. Even if a detector has a ratio of 1, it might not detect everything; it only means all its attempted detections are correct.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}. \tag{4.1}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegatives}. \tag{4.2}$$

Recall is the measure of false negative rate. It measures the ability of the detector to find all the positives. An example would be being able to find

60% of the amount of cats present in an image. These factors are directly dependant on each other to function, as well as the threshold set. Threshold is the confidence score the detector must cross before an object is positively detected. A high threshold means when guessing, it is usually correct but misses a lot of other objects present (high precision, but low recall). On the other hand, a low threshold does the opposite; detects everything, but inaccurately. Hence, the threshold set greatly affects the results. Finally, the AP score is created by taking the average precision value across all the recall values. The exacts mathematical equation differs, depending on tests used (COCO, PASCAL VOC) [7].

To understand localization measures, Intersection over Union is used (IoU). In short, it measures how well the predictions overlaps with the ground truth box. Moreover, the mAP score is calculated by taking the mean AP of all classes over all the IoU thresholds. COCO tests can be seen as mAP@[.5:.95], meaning they go from an AP with an IoU of 50% overlap to 95% overlap. The average of these is what creates the mean average precision (mAP), which is the measure used by many detectors today [28] [29]. This does however only tell how accurate a detector is, not its speed.

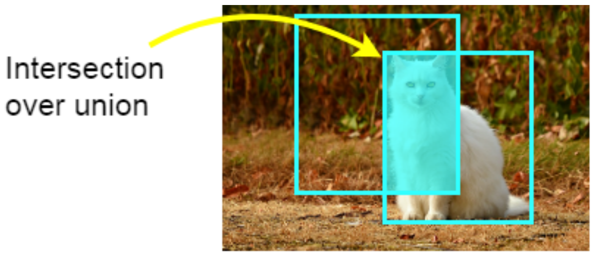


Figure 4.1: IoU of a cat.

4.1.2 COCO Benchmarking Test

To have some sort of benchmark in regards to performance, a comparison to other existing implementation is done. However, there are very few implementations, even lesser with stated results. To rather have a reliably understanding of its performance, three implementations have been used as benchmark; A Chainer implementation, a Pytorch implementation and the original paper. It is worth mentioning every implementation is done in regards to the original paper. However, as the public code of Lighthead R-CNN is far too specific to their case of testing, recreation creates unique results, which can be seen from various people trying to recreate the Lighthead R-CNN results.

Furthermore, these implementations usually focus on either speed or accuracy, not a trade-off in between. The aim of the project is to optimize accuracy and achieve an acceptable frame-rate at the same time, not either or. Further, it is better for such a problem to trade off higher FPS, such as 30, in the favor of accuracy gain. As such, a certain bias may be present as the approach of existing solution might not be optimized to the scenario in question, while the approach taken in the project might not surpass results achieved by other implementations aiming to optimize specifically accuracy or speed.

Lastly, the results provided by the different implementers have to a certain degree be trusted, as recreating every implementation is not feasible timewise, as well as the availability of hardware in regards to what each approach have used.

The original paper primarily shows two results; one for a high accuracy and one for speed. The speed of the network is at 102 FPS with an mAP of 30.7 (224×224 images) [1], performed on the COCO 2014 dataset test. The approach uses an Xception-like backbone, essentially trading accuracy for speed. The accuracy approach uses Resnet101 with an mAP@[0.5:0.95] of 40 (tested on trainval 2014) but an unreported speed at the level of accuracy. The prediction and training is done on 8 Nvidia Titan XP [21], hence the inference time at 102 FPS cannot be compared to the availability of hardware in this project (implementation on a Nvidia 1080 Ti) . Hence, it is hard to use it as a benchmark but more as a guideline as the project is based off the algorithm. However, having an mAP@0.5:0.95 between 30-40 while still achieving 7-10 FPS means beating the accuracy of an original Xception implementation while still achieving the speed requirements of the task.

The algorithm was trained for 30 epochs on a Nvidia 1080 Ti on the same datasets as other Lighthead R-CNN implementations, COCO testval 2014 (80.000 images for training, 35.000 for validation, 5.000 for testing). The result of training iterations can be seen in figure 4.2. Testing on newer datasets could yield different results, hence the testing on old data for a proper comparison. Results and comparisons are shown in table 4.1.

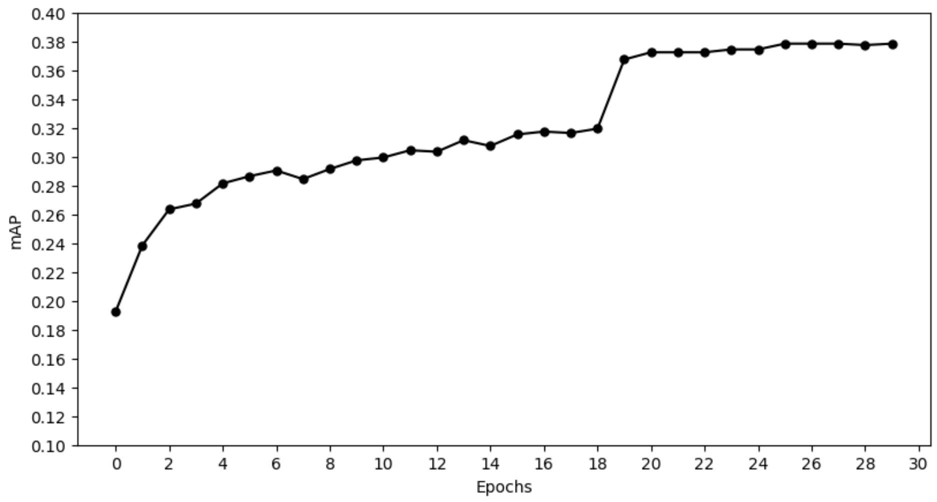


Figure 4.2: Evaluation of training, 30 epochs, 1000 × 800 image size.

Implementation	**mAP@0.5:0.95**	mAP@0.5	mAP@0.75
Original	0.400	0.621	0.429
Original with Xception	0.307	-	-
Implementation in Pytorch	0.3963	0.601	0.432
Implementation in Chainer	0.391	0.607	0.419
Ours	**0.379**	0.589	0.407

Table 4.1: Comparison of accuracy on a COCO test to other implementations.

Both the pytorch and chainer implementations have similar results to the original, albeit somewhat lower accuracy. The implementations proposed learns faster than the other implementations when tested on the same dataset (the dataset created in the project). It is not as accurate as the other implementations, albeit that is not the aim; it is to be faster to achieve acceptable inference time while still having a higher mAP than when changing to a lighter backbone. The accuracy loss in mAP is acceptable at the increase of speed achieved as it is just behind the other implementations focusing solely on a high mAP and well above implementing lighter backbones.

As stated earlier, the network is optimized for a framerate of 7-10 FPS, which it achieves with an mAP of 0.379 at an image size of 1000 × 800. However when having a live video feed, additional images are fed into the network as the rate of images sent are bound to external software and hardware. Hence, a queue arise as the inference speed is lower than the rate of image arrival and a delay incur. There is a limit to the delay, albeit the aspect of real-time is lessened. Hence, in the two tests below image size is reduced 640 × 480 to increase the FPS and reduce the delay. Naturally this affects the mAP@[0.5:0.95] which drops down to 0.339, which is still above using other lighter backbones. As such, a theoretical max mAP is 0.379 but because of external limitations some of it has to be sacrificed for additional speed to compensate for delay.

Comparing the speed of different implementations is somewhat difficult; first off there could be errors in reimplementation. Second, different hardware plays a major role. If looking at the inference time of the original Lighthead through graphs, it has an inference time of about 125 ms [1]. Directly this means 8 FPS, but other forms of delay in real-time exist than pure prediction of images. Transferring of data back an forth between an external camera to a predictor, rendering of boxes and other similar issues not related directly to prediction takes time. Further, the original implementation had 8 Pascal TITAN XP GPUs at their disposal, whereas this project has one 1080 Ti GPU but still achieving 7-10 FPS in real-time prediction. A recreation of the original algorithm without modification on the 1080 ti had around 1 FPS, but errors in reimplementation can have occurred. The same goes for the other implementations, achieving roughly the same speed as the original but somewhat less mAP (albeit the aim of these were never speed, but accuracy). The implementation proposed in the thesis is faster, but at the cost of accuracy compared to other implementations. However, this is ideal when faced with the problem the project aims to solve.

4.1.3 Evaluation of Steps and Detection

This section is about tests regarding the detection of individual components. Its purpose is to see how well the solution performs in regards to the specific problem faced. Further, it allows for unveiling issues, errors and weaknesses of the proposed solution before it is tested on real users. This is done preemptively to the last test to avoid unnecessary problems, improve upon some annotations and weaknesses where required. Ultimately, it is done to achieve optimal results of the final test which is the main focus of the project.

It is expected to film the whole object and not too partially. Further, filming too close to an object might create obscurities. Hence, a user is told to film at reasonable lengths depending on the object. Lastly, varying parts are tested on; some included in the training data (even though the data set used uses many different instances of the same class), some which have similarities and some which are different. This is to avoid too much bias, although most computer parts share very many similarities. Two different CPUs are almost identical, the same goes for RAM, SSD, HDDs and so forth.

Motherboard
The initial prediction is the motherboard, which it accurately predicts at 80-99% certainty. No other classes of motherboard is predicted, albeit only three are present total. A large amount of motherboards could cloud this part, although additional training data would counter this issue. The detection starts around a height of 70 cm down towards 25 cm before it becomes inaccurate. Its detection angle is between 45 to 135 degrees. The initial threshold of 40% is pushed to 60% to avoid false positives.

RAM
The detector starts detecting around 35 cm and gets better the closer it gets, with a certainty between 70-99%. It does not detect false positives, but cannot reliably detect the difference between DDR4 and DDR3 when it is at max distance and when both RAM pieces are green. This is most likely caused by the main source of DDR3 being without an additional layer of protection, whereas many of the DDR4 have this. The main difference is somewhat subtle and is hard to distinguish at a distance. However, the issue perish when filmed close and the entire piece is in sight. Angle is not important to detect RAM itself, but compatibility needs to be directly above to see the distinguishing feature. Threshold is set to 40%.

RAM Slots
RAM slots are one of the best performing detections, with a certainty of 95-99% between 0-180 degrees. There are no false positives at a threshold of 40%. It starts detection at 40 cm height, this could be stretched even further by lowering the threshold albeit false positives could incur. Moreover, almost no other class has this level of height detection, rendering such a change useless.

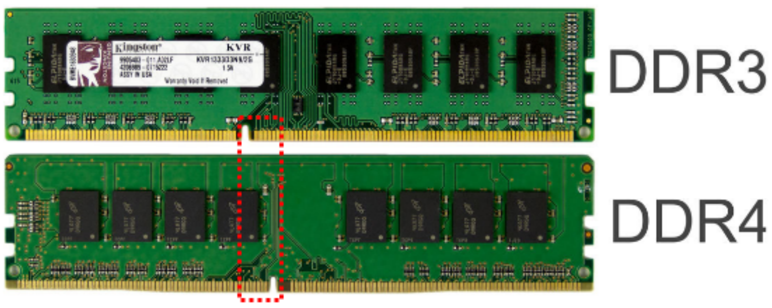


Figure 4.3: Difference between DDR3 and 4 [30].

CPU
The CPU starts getting reliably detected at a height of 20 cm and below with a certainty of 90-99%. It does so from both sides and almost any angle with no false positives. Interestingly enough, the CPU socket was not considered a CPU from the golden side, even though they share many similarities. Threshold is set to 40%, as it needs to be somewhat close before it starts detecting.

CPU Socket
CPU socket starts detection at 30 cm and below with a certainty between 80-99%. There are no false positives and can be detected from many angles. Threshold is set to 40%.

CPU Fan
The CPU fan is detected at 35cm and below with a certainty between 40 and 99%. At a threshold of 40% there is one false positive, the PSU. This is most likely caused by the fan itself, as it is only falsely detected at the specific angle where the fan is visible on the PSU. To counter this, threshold is adjusted to 60%.

CPU Fan Slot
CPU fan slot starts detection at 30 cm and below with a certainty between 60-99% with a max angle between 45-135 degrees. There are no false positives at a threshold of 40%.

CPU Fan Power Slot
CPU fan power slot has no false positives and starts its detection at 20 cm height, with a certainty between 40-99%. Multiple angles are allowed for with a degree between 20-160 and a threshold at 40%.

Fan Power Cable
The fan power cable is detected at 20 cm and below, but usually needs to be somewhat close with an ideal angle to reliably detect. It has a certainty between 40-99%, but has one false positive at specific angles (the 6-pin cable).

PSU
The PSU has no false positives, but struggles at times when filmed directly from above. Hence, a slight tilt is required to detect at 50% threshold, which detection then start at 40 cm with a certainty between 40-99%.

6 and 8-Pin Power Connector GPU
The 6-pin connector is easily detected, but it grabs some part of the 8-pin connector as well. This is expected as parts of the 8-pin are identical to the 6-pin. It struggles a bit more to detect the 8-pin and needs to be close, around 5-10 cm with a good lighting and angle. It the certainty bounces between 10-99%, being one of the worse parts. It does not have false positives other than the issue above, but this is of lesser consern as both cables goes next to each other on the GPU. A threshold is set at 40%. It shows that cables are hard to detect and requries close-up with proper angles to correctly predict.

8-Pin Power Connector CPU
Much like above, the 8-pin CPU power connector requires close-up to properly detect (around 5-10 cm). It does not usually have false positives but struggles when black cables of bad angles are used. This is most likely caused by the lack of training on the class, which has only a total of 10 instances. At good angles and close range the certainty is above 90%, but at bad angles it cannot be detected even at a threshold of 10%. Threshold is set to 40% to avoid false positives, but additional instances in training are added to compensate for the bad accuracy. The current implementation detects both 8 pin CPU cables and 4+4.

GPU
The GPU starts to be detected around 30-35cm height with a certainty of 70-99%, depending on the side scanned. It is usually best detected when the larges sides of the GPU is shown, not its thin sides. If one zooms into a green RAM DDR4 piece, it can be detected as a GPU at a very specific angle. However, the certainty is low and a user would normally never film at this angle and range, essentially not making this a problem.

PCI-express x16
No false positives at any angle and height, which it starts to detect around 30 cm. When detecting, it always has a certainty between 90-99%. Angle of detection can be from 0 to 180 degrees and threshold is set to 40%. This is the best class detected of all, likely because of the amount of instances compared to the rest. This goes to show increasing the training data further can potentially enhance the rest of the classes as well.

6 and 8-Pin Power Slot GPU
The GPU 6 and 8-pin power slots have no false positives. The 6-pin is easily detected, although the same issue from earlier persist; detecting parts of the 8-pin as a 6-pin. Again, this is not significant as both cables are going next to each other on the same device. Detection starts around 20 cm and below with a certainty between 60-99%, although it needs good lighting. The threshold is set to 40%.

24-Pin Power Connector
The 24-pin power connector cable is the most easily detected cable with a certainty between 40-90% and detection range from 20 cm. There are no false positives at 40% threshold, but is usually best detected from above, not directly into the pins themselves. Hence, some angulation is required to detect but far less than other cables.

24-Pin Power Connector Slot
The 24-pin power connector slot starts detection at around 25 cm height, able to be detected at many angles with a degree of 20-160 with a certainty between 70-99%. Threshold is set to 40%, which it has no false positives.

8-Pin Power Connector Slot CPU
Height of 30 cm at first detection and angulation of 60-120 degree with certainty between 70-99%. It is normally extremely stable at every angle and with zero false positives at 40%.

SATA Cable
The SATA cable, being of the cable category, needs to be very close to the camera before being detected. It starts off at 10 cm and gets better when closer, with a certainty between 50-99% but will only detect at certain angles. It also predicts a cable being SATA regardless of the variant (some are bent, some are straight). A false positive for SSD/HDD power cables can occur at times from a specific angle, as it can look identical when tilted correctly. In practice this is not an issue, as both cables go next to each other

on a HDD or SSD. Some other cables which have similarities to a SATA can be falsely detected, hence the threshold is raised to 80% to reduce false positives. However, SATA is not bound to the PSU. As such, guiding tips are shown on screen to the user to not look for cables tethered to a PSU but for external, individual cables. The detection ensures the user they have the correct cable.

SATA Slots (internal)

While SATA slots are identical on both SSD/HDD and the motherboard, they are split into two (internal and external) when testing. This is to reduce false positives, as well as better guidance of a user to reduce confusion. On internal, it detects at 20 cm and below, with a certainty between 70-99%. At 40% threshold some false positives occur and thus it is pushed to 95%. It needs to be filmed somewhat closer with this threshold, but its detection is pretty clear when detected.

SSD and HDD

The SSD and HDD are both detected at a height of 30 cm and below with a certainty between 70-99%. At times it can mix a HDD and SSD when the metallic side of the HDD is shown, as it is identical in color to some SSDs. However, this certainty is much lower and when a HDD is flipped showing its lesser metallic side this issue cease to exist. This is not really an issue as both components uses the same cables, however the threshold is raised to 70% to reduce the false positive rate.

HDD/SSD SATA Slots (external)

As stated above, this is the external SATA slots found on the SSD and HDD. It is detected from 15 cm and below, between 70-99% certainty. As earlier, it can mix the SSD/HDD power slot and SATA slot at certain angles at lower thresholds as they can look identical. This can be avoided by increasing the threshold, albeit this is not necessary as both go next to each other on the component and a user will most likely be able to distinguish this themselves when holding the corresponding cables in hand and told the general location of where to put said cables. Hence, the threshold is set to 40%.

SSD/HDD Power Cable

The power cables to SSD and HDD is detected around 5-10 cm and below, between 40-90%. It needs to be close to the camera and in focus to detect at all, although this goes for many of the cables. Much like the SATA cable, it has false positive as a SATA cable at specific angles. As such, the threshold is set to 70% to counter this.

SSD/HDD power slot
The power input on the SSD/HDD is detected at 5-10 cm and below, between 70-99% certainty. It does at times require adjustment of angle to detect, but has no false positives. The threshold is set to 40%.

Summary
To summarize the tests, the internal components are almost a non-issue. Furthermore, most of the external parts are easily detected as well, but might require slight tilting of a camera to properly detect. Cables are definitely the most challenging category to detect. While parts of this is caused by a lower amount of training data, it is also caused by the level of detail present in combination with lower resolution images. As a result of this, additional training data is added to where it is required to reinforce the detection. Further, written tips are added at certain steps where errors may occur. These are simple but effective and important tips, such as telling a user to try different angles, vary the distance and try more components if he/she is unsure. When the detector finds the object of interest it is usually pretty sure, but false positive can occur as a flicker, which is not the case with true positives. As such, it is possible to reliably detect components and sockets on a motherboard including compatibility checks, but it requires a user to be aware of how to use the detector correctly.

4.2 Tests on Real Users

This section covers the experiment on real users, where persons with low technical skills are given all the pieces required to assemble a computer as well as the AI to assist them. While no information or guidance on how to assemble a computer is given directly, tips and guidance on how to use the detector is initially given. This is not orally given but textually through the application itself. For example, it explains the difference between internal and external parts as this is a crucial part of the design of the system. Further, it explains the ideal distance to scan at depending on what is being scanned, to rotate an object and the camera if nothing seems to be detected or try different components if a user is uncertain, as the detector is usually very certain when the correct object of interest is at an ideal angle. As for fitting, a user will have to align and put the components in place themselves but with backup in forms of textual tips according to the step a user is at.

The test itself is performed to see if implementing such a solution can support human beings of little previous experience with computer assembly. The other tests are performed to understand the system's capabilities and weaknesses and improve upon these prior to this test to obtain better results. Users are tested individually without any human assistance with the goal of assembly from start to finish. The goal is not to connect external components such as a monitor or mice but assemble all the components onto a motherboard itself. Evaluation is performed by two other individuals experienced in computer assembly. There is a difference between errors as well; Non-critical errors or error-free rate include struggles with fitting, flickers of false positives and other related issues which does not stop a user from finishing the assembly correctly. Critical errors include issues of zero detection of an object of interest or a user simply not understanding on how to execute a step, leading to failure in assembly. This is referred to as the completion rate. Time is noted, but not seen as a critical parameter. The solution was tested on a total of 13 people, all of similar technical background.

Summarized, the completion rate of the user tests are at 95.99% with an error-free rate of 86.54%. The average time of the steps accumulated is 29 minutes, albeit this does not include the amount of time spent in between explanatory black screens which some users spent considerable time in, resulting in a total time spent in assembly exceeding just around an hour. This however, varied a lot. A video of the guide and images of some of the participants in the assembly process can be seen in the link.[1]

Several interesting observations were made throughout the tests; Fitting seemed to be an issue for some users, as they were often afraid of a component being fragile and not using enough force to push a component or cable into place. Some simply forgot to screw or properly fit components, resulting in the component in question falling or being loosely fit. The compatibility check worked as intended with one exception with DDR4, were a user fit the RAM card between two RAM slots as it did not fit into the DDR3 slot. At certain steps where users were stuck, they had to be reminded to read the tips carefully. However, this led to reduction in the error-free rate. An interesting part of the test was the increasingly difficulty in detecting the

[1] `https://www.youtube.com/watch?v=B1MpH66SGIM`

Step	Completion rate	Error-free rate	Time in seconds
Motherboard check	100	100	41.41307692
RAM detection	100	100	26.86384615
RAM slot detection and fitting	84.61538462	61.53846154	93.11769231
CPU detection and compatibility check	100	100	40.97153846
CPU socket and fitting	100	100	29.10230769
CPU fan	100	100	23.64076923
CPU fan slot and fitting	100	84.61538462	255.7692308
CPU fan power slot	61.53846154	46.15384615	127.4615385
Fan power cable and fitting	80.76923077	46.15384615	26.15384615
PSU	100	100	14.92307692
6 and 8-pin power connectors GPU	76.92307692	61.53846154	51.73230769
GPU detection	100	92.30769231	43.92307692
PCI-E 16 detection and fitting	100	100	189.0769231
6 and 8-pin power slots (GPU) and fitting	100	100	41.37615385
24-pin power connector	100	84.61538462	44.72615385
24-pin power slot and fitting	100	100	29.07692308
8-pin power connector (CPU)	100	84.61538462	23.85153846
8-pin power connector (CPU) and fitting	100	100	39.28384615
S-ATA cables	100	100	22.15
SSD/HDD	100	100	32.97076923
SSD/HDD S-ATA input and fitting internal	100	53.84615385	264.3846154
SSD/HDD S-ATA input and fitting on SSD	100	69.23076923	189.4615385
SSD/HDD power cable	100	92.30769231	52.30769231
SSD/HDD power input and fitting	100	100	37.36692308

Table 4.2: Steps with the completion rate and error-free rate in percentage as well as time. The numbers are an average over 13 different participants.

correct object in question the further a user got into the guide; as more parts are added onto the motherboard, the harder it gets to properly detect the correct object. For the most part this was a non-issue and only resulting in the participant using additional time to complete the current step. The main observation made in the test was the variance of users. Even though every participants had about the same amount of technical understanding, the use of the guide varied a lot. Some heavily relied on the AI and its detection, entirely skipping the tips. This resulted in redundant scanning and an increase in time used. Others did not seem to want to use the detector and tried trail and error, also resulting in a larger time used than others. The best group seemed to be the participants using a balance of the detector, tips present on each step and thinking for themselves as well (i.e. a large card cannot fit into a small 8-pin socket). The detector and its performance was not an issue, but rather its usage. Generally, participants were able to assemble a computer correctly with the assistance of the detector, albeit the time usage varied greatly.

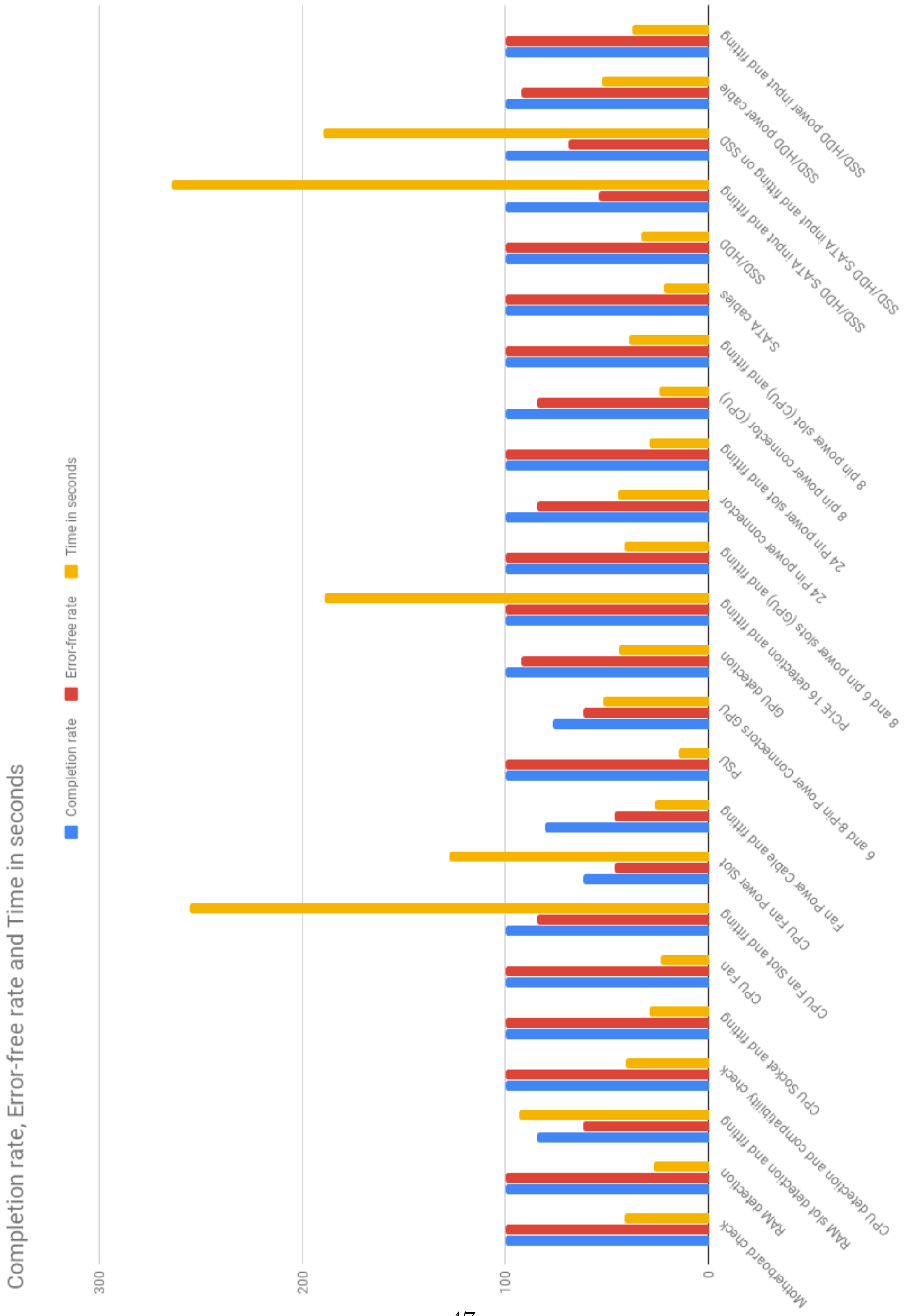


Figure 4.4: Summary of 13 users across every step.

Chapter 5

Conclusion and Future Work

This chapter concludes the thesis, discusses issues and talks about potential future work for improvement.

5.1 Conclusion

The problem of AI assisting humans in computer assembly has been studied in this work. To make a proper conclusion, hypotheses from the start are checked. From the detection test, it is shown that an AI can indeed reliably detect computer components and sockets on a motherboard. Further, through modification and alteration to the existing algorithm, it is usable in real-time on realistic hardware, such as a singular Nvidia 1080 ti rather than 8 Titan X while still achieving an mAP of 37.9%. Seen in the experiments on real users with lesser technical understanding, artificial intelligence can indeed not only assist, but also be completely relied upon to assemble a computer. The approach led to a completion rate of 95.99% and an error-free rate of 86.54%. However, while the detector itself is not an issue, the extent of replacing a vendor is arguable. The assembly of the computer itself is indeed possible, albeit a casing is usually preferred as well which is not something that the approach takes into consideration. Furthermore, a powerful external computer is required to process the images albeit this can be outsourced using edge computing and 5G networks.

The work has been successful in verifying artificial intelligence's ability to assist humans with little to no technical background in computer assembly. Given the computational power, other relevant algorithms and implementations were not able to fulfill the task at hand caused by their lack of speed in favour of accuracy. The solution proposed proves its viability as an option to having a pre-built computer from a vendor or following online video guides. It can also be seen as a stepping stone towards the self-reproduction of AI by making new computers.

5.2 Discussion and Future Work

In hindsight, there are additions and other choices that could have been made which could potentially improve the solution proposed. Firstmost is the aspect of the detectors performance, in which there are several potential improvements. To reduce the delay introduced by the camera, the render-time through OpenCV can be improved by rewriting parts of the library used, allowing for a decrease in delay when filming [31]. Another choice with more potential impact is the change to a new backbone, a promising one with interesting results is shufflenetv2. However, this would mean restructuring the entire network, requiring significant amount of time. Further, there is a lack of evidence if introducing said backbone will actually result in improvement and thus requires further testing. Lastly, additional training data would most likely improve the detection of components where the detector struggles, such as cables. Proper lighting and good focus is always required, although it could potentially be compensated by additional training data.

From the final test, several points of improvement to be made were revealed. A common factor for all participants were the fact they did not like filming on one device and looking at another screen for detection. This can have had a negative impact on the results obtained in the test. Moreover, some users did not like following a guide and simply ignored what to do at certain points. It seemed the AI itself was rarely the issue but rather users not using the detector or tips at all. Furthermore, issues as simple as forgetting to screw a fan or putting a cable in place occurred. Improving the AI does little when an error is mostly human-side. As such, an improvement could be made to the textual explanation of the guide itself, be it graphical or textual. The variation between participants were large at times, indicating external human factors having a larger impact on the end result than expected. As such, a larger test-base would be ideal to lessen the impact individual persons can have on the test.

References

[1] Zeming Li et al. “Light-Head R-CNN: In Defense of Two-Stage Object Detector”. In: *CoRR* abs/1711.07264 (2017). arXiv: 1711.07264. URL: http://arxiv.org/abs/1711.07264 (visited on February 23, 2019).

[2] Giles Westerfield, Antonija Mitrovic, and Mark Billinghurst. *Intelligent Augmented Reality Training for Motherboard Assembly.* 2015. URL: https://link.springer.com/content/pdf/10.1007%5C%2Fs40593-014-0032-x.pdf (visited on March 1, 2019).

[3] Jonathan Hui. *Design choices, lessons learned and trends for object detections?* 2018. URL: https://medium.com/@jonathan_hui/design-choices-lessons-learned-and-trends-for-object-detections-4f48b59ec5ff (visited on February 23, 2019).

[4] Jonathan Huang et al. “Speed/accuracy trade-offs for modern convolutional object detectors”. In: *CoRR* abs/1611.10012 (2016). arXiv: 1611.10012. URL: http://arxiv.org/abs/1611.10012 (visited on February 23, 2019).

[5] Deep AI. *Feature Extraction.* 2018. URL: https://deepai.org/machine-learning-glossary-and-terms/feature-extraction.

[6] Stanford Vision Lab. *ImageNet.* 2016. URL: http://www.image-net.org/ (visited on February 26, 2019).

[7] COCO Consortion. *Common Objects in Common Context.* 2018. URL: http://cocodataset.org/ (visited on February 25, 2019).

[8] Dutta A., Arandjelovic R., and Zissermann A. *VGG Image Annotator.* 2016. URL: http://www.robots.ox.ac.uk/~vgg/software/vise/ (visited on February 25, 2019).

[9] Lilian Weng. *Object Detection Part 4: Fast Detection Models.* 2018. URL: https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html (visited on February 28, 2019).

[10] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870 (visited on February 26, 2019).

[11] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO—Object Detection Algorithms.* 2018. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (visited on February 26, 2019).

[12] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.106. URL: http://dx.doi.org/10.1109/CVPR.2017.106 (visited on February 26, 2019).

[13] Vahid Mirjalili. *How does the region proposal network (RPN) in Faster R-CNN work?* 2018. URL: https://www.quora.com/How-does-the-region-proposal-network-RPN-in-Faster-R-CNN-work (visited on February 28, 2019).

[14] Jeremy Jordan. *An overview of object detection: one-stage methods.* 2018. URL: https://www.jeremyjordan.me/object-detection-one-stage (visited on February 28, 2019).

[15] Joseph Redmon. *pjreddie yolo website.* 2018. URL: https://pjreddie.com/darknet/yolo/ (visited on March 1, 2019).

[16] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection.* 2016. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf (visited on March 1, 2019).

[17] Hao Gao. *Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch.* 2018. URL: https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad (visited on February 28, 2019).

[18] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767 (visited on March 1, 2019).

[19] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *CoRR* abs/1708.02002 (2017). arXiv: `1708.02002`. URL: `http://arxiv.org/abs/1708.02002` (visited on March 3, 2019).

[20] Ayoosh Kathuria. *What’s new in YOLO v3?* 2018. URL: `https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b` (visited on March 1, 2019).

[21] Zeming Li et al. *Light-Head R-CNN: In Defense of Two-Stage Object Detector.* 2018. URL: `https://github.com/hwkim94/hwkim94.github.io/wiki/Light-Head-R-CNN:-In-Defense-of-Two-Stage-Object-Detector(2017)` (visited on March 4, 2019).

[22] Hayabusa. *ChainerCV and Light-Head R-CNN” Camera / Movie Compatible.* 2018. URL: `https://cpp-learning.com/chainercv_light-head-r-cnn` (visited on March 4, 2019).

[23] Hayabusa. *Object detection by ChainerCV and Light-Head R-CNN.* 2018. URL: `https://www.youtube.com/watch?v=dv3MA1Sv2ew` (visited on March 4, 2019).

[24] Ningning Ma et al. “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design”. In: *CoRR* abs/1807.11164 (2018). arXiv: `1807.11164`. URL: `http://arxiv.org/abs/1807.11164` (visited on March 5, 2019).

[25] François Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”. In: *CoRR* abs/1610.02357 (2016). arXiv: `1610.02357`. URL: `http://arxiv.org/abs/1610.02357` (visited on March 5, 2019).

[26] WorldEyeCam. *FPS comparison video 1fps 7fps 15fps 30fps.* 2015. URL: `https://www.youtube.com/watch?v=IgdOupqPi5c` (visited on March 25, 2019).

[27] Keras. *Documentation for individual models.* 2018. URL: `https://keras.io/applications/` (visited on March 3, 2019).

[28] Jonathan Hui. *mAP (mean Average Precision) for Object Detection.* 2018. URL: `https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173` (visited on March 26, 2019).

[29] Timothy C Arlen. *Understanding the mAP Evaluation Metric for Object Detection.* 2018. URL: `https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3` (visited on March 26, 2019).

[30] Matt Bach. *Tech Primer: DDR4 RAM*. 2014. URL: `https://www.pugetsystems.com/labs/articles/Tech-Primer-DDR4-RAM-589/` (visited on April 11, 2019).

[31] Adrian Rosebrock. *Faster video file FPS with cv2.VideoCapture and OpenCV*. 2017. URL: `https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/` (visited on April 27, 2019).

Appendices

A Hardware Specification

Operating System	Ubuntu 18.04
Processor	Intel i7-8700K 3.70 GHz
Memory	16GB DDR4
Graphics	1 × NVIDIA GeForce 1080 ti

B Class Description

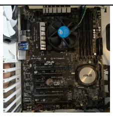
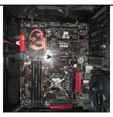
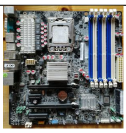
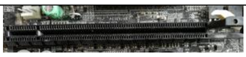
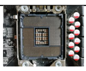
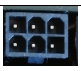
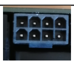
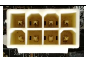
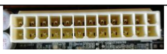
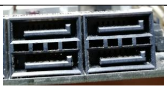
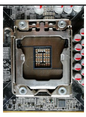
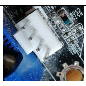
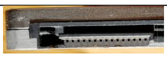
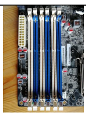
Part	Images	Note
ASUS Z97-AR (motherboard)		
MSI Z87-G45 Gaming (motherboard)		
Gateway TBGM-01		
PCI-express x16		
CPU socket		
6-pin power connector slot		
8-pin power connector slot GPU		
8-pin power connector slot CPU		
24-pin power connector slot		
S-ATA slot		
CPU fan slot		
fan power slot		
SSD/HDD power slot		
RAM slots		DDR 3 and 4

Table 1: Internal parts

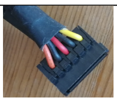
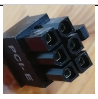
Part	Images	Note
SSD		
HDD		
DDR3 RAM		
DDR4 RAM		
SSD/HDD power cable		
GPU		
CPU		
CPU fan		
CPU fan power connector		
PSU		
6-pin power connector		
8-pin power connector GPU		
8-pin power connector CPU		
24-pin power connector		
S-ATA cable		

Table 2: External parts