

Development of a Semantic Model and  
Synthetic Dataset for Multi-Grasp  
Affordance Detection for Application to  
Vision-Based Upper-Limb  
Prosthetic Grasping

by

Nathan Yin Hang Ng

A thesis  
presented to the University of Waterloo  
in fulfilment of the  
thesis requirement for the degree of  
Master of Applied Science  
in  
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2024

©Nathan Yin Hang Ng 2024

## **AUTHOR'S DECLARATION**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

## Abstract

Current upper-limb prosthetic grasping methods are predominately myoelectric, where surface electromyogram (sEMG) pattern recognition is used to predict a grasp type for a prosthetic hand to grasp objects. The sEMG patterns also simultaneously detect the action intent of a grasping action and overall movements of the prosthetic arm. Since the overall control strategy of a myoelectric prosthesis is coupled, the prediction of grasp types can be inaccurate, especially if the grasp type has a similar sEMG pattern for manipulating the prosthetic arm or selecting other grasp types. Recent vision-based prosthetic grasping methods solve the coupled control strategy of myoelectric prostheses, by implementing a camera system to capture an RGB image of an object and a convolutional neural network (CNN) to predict a grasp type. The action intent to move the prosthetic arm and perform the grasping action is independently determined through sEMG pattern recognition. Unlike myoelectric prostheses, vision-based prostheses can predict a suitable grasp type based on the features of an object (e.g. object's shape). However, current vision-based grasping methods are limited because each object can only be grasped with a single grasp type, despite the object's shape, environmental context, and the available tasks. Recent robotic grasping applications implement grasp affordance detection to identify the regions on an object that can be grasped for a task. By adapting the detection of grasp affordances into a vision-based prosthetic device, multiple task-oriented grasp-type predictions are possible for each object. Therefore, to improve the vision system in vision-based prostheses, grasp affordance detection methods from robotic grasping applications are adapted in this thesis research.

Grasp affordances, as grasp-type and task regions, are predicted by implementing instance segmentation models. Instance segmentation models utilize RGB images to localize objects and their grasp affordances with bounding box locations and image mask segmentation. Since there is no instance segmentation model and dataset that can allow the simultaneous detection of objects and their grasp affordances, the Multi-Affordance Detection Network (MAD-Net) model and Multi-Object Multi-Grasp-Affordance (MOMA) synthetic dataset were developed as part of this thesis research. Unlike the current vision-based prosthetic grasping methods, MAD-Net can detect objects and their grasp affordances in multi-object RGB scenes. The MAD-Net model was derived from the Mask R-CNN model, a common baseline model for instance segmentation. Most instance segmentation models were derived from Mask R-CNN, since the additional mask prediction head in Mask R-CNN can convert all object detection models into instance segmentation models. The MOMA synthetic dataset is a collection of 20K RGB images that is generated from placing random images of objects on random background images. Each

image generated was automatically annotated with the instances of objects and their grasp affordances (grasp-type and task regions). The single-object RGB images used for synthetic dataset generation were manually captured with a camera and then manually annotated.

The mean average precision (mAP) metric is used to evaluate the performance of MAD-Net and other instance segmentation models on the MOMA dataset. The mAP metric is a good indicator of model performance, since it determines how accurate the predicted bounding box and image mask locations are w.r.t. the ground truth annotations. MAD-Net has outperformed all the other instance segmentation models across all detection categories (objects, grasp types, tasks) on the validation datasets. On the test datasets, MAD-Net has maintained a similar mAP score as the other instance segmentation models. In all cases, MAD-Net has outperformed Mask R-CNN, especially in the grasp-type detection category, where MAD-Net has a 10% increase in the mAP score compared to Mask R-CNN. When the objects and their grasp affordances are jointly trained on the MOMA dataset, the total training time decreased by 50%. Since MAD-Net has outperformed Mask R-CNN, the joint detection of objects and their grasp affordances is a feasible solution to implement in the vision system for vision-based prostheses. Although the proposed vision system produces multiple task-oriented grasp types on a single object, modern myoelectric prostheses can select a grasp type from a small selection of pre-programmed grasp types. A grasp database can also be implemented alongside the proposed vision system. Prosthetic users can continuously update the database for new unseen objects and their corresponding task-oriented grasp types.



## Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Jonathan Kofman, for his everlasting support, patience, and professional guidance throughout my journey in the master's program. Without Dr. Kofman, I would not have been able to kickstart my dream career in machine learning and computer vision.

I would like to thank my co-supervisor, Dr. Soo Jeon, who has also patiently supported me throughout my master's journey and provided professional guidance in refining my research direction. I would also like to thank Dr. Jeon for his teachings in modern control systems.

I am honoured to have Dr. James Tung and Dr. Zhao Pan to serve as my thesis readers. Thank you for all the insightful feedback you have provided to improve my thesis.

Thank you to my friends and labmates, Lukas Stracovsky and Ali Asghar, who were also like a mentor to me. I appreciate the weekly introduction of new concepts and meaningful discussions in the meetings. I would also like to thank Lukas for providing me with the opportunity to remind him of upcoming important events.

Finally, I would love to give an extended thanks to my family, especially my mother and brother, who have continually refuelled my motivation to succeed in my master's program.

## **Dedication**

This thesis paper is dedicated to my family, especially my mother, Yvonne, and brother, Marco. Without their continuous support, my journey in the master's program would have been a disaster.

I would also like to dedicate my thesis to my undergraduate professors at the Ontario Tech University, Dr. Amirhossein Monjazebe and Dr. Yuelei (James) Yang. Their passion in teaching robotics has built the very core foundations of my career in robotics and mechatronics engineering. Dr. Monjazebe and Dr. Yang have also helped and supported my decision in pursuing my master's degree at the University of Waterloo.

# Table of Contents

AUTHOR'S DECLARATION.....	ii
Abstract.....	iii
Acknowledgements.....	v
Dedication.....	vi
List of Figures.....	x
List of Tables.....	xii
Chapter 1 Introduction.....	1
1.1 Visual Context and Affordances.....	1
1.2 Types of Commercially Available Upper Limb Prostheses and Their Limitations.....	2
1.2.1 Body-powered Prostheses.....	2
1.2.2 Myoelectric Prostheses.....	5
1.3 Vision-based Prosthetic Grasping and Related Vision-based Methods in Robotic Grasping.....	6
1.4 Research Objectives.....	8
1.5 Research Contributions.....	8
1.6 Thesis Outline.....	9
Chapter 2 Background and Literature Review.....	10
2.1 Definition of a Grasp.....	10
2.1.1 Prehensile and Non-Prehensile Grasps.....	11
2.2 Grasp Taxonomy.....	12
2.2.1 Grasp Type Recognition in Human Activity Datasets.....	15
2.3 Grasp Estimation and Representation.....	16
2.3.1 Grasp-Type Classification and Detection for Grasp-Pose Estimation.....	18
2.3.2 Object-Pose Estimation in Grasp-Pose Estimation.....	19
2.3.3 Applications of Hand-Pose Estimators in Grasp-Pose Estimation.....	21

2.4 Affordance Detection .....	23
2.4.1 Object Affordance Detection .....	24
2.4.2 Grasp Affordance Detection.....	27
2.5 Task-Oriented Grasping .....	27
2.6 Summary .....	28
Chapter 3 Methods for Simultaneously Detecting Objects and Their Grasp Affordances with A Deep CNN Segmentation Model.....	31
3.1 Ablation Study to Determine the Significance of Object Localization in Instance Segmentation Models .....	32
3.1.1 ResNet-34 FCN Semantic Segmentation Model Architecture .....	32
3.1.2 Training and Evaluation of the ResNet-34 FCN Semantic Segmentation Model.....	34
3.1.3 Results and Discussion of the ResNet-34 FCN Semantic Segmentation Model.....	38
3.2 Preparation of Single-Object RGB Images for Synthetic Dataset Generation.....	42
3.2.1 Image Collection of Real Objects in Single-Object RGB Scenes .....	43
3.2.2 Image Collection of Synthetic 3D Objects in Single-Object RGB Scenes .....	47
3.2.3 Preparation of the Single-Object RGB Images for Object Mask Annotations.....	48
3.2.4 Annotation of Grasp Affordances as Instances of Image Masks .....	53
3.2.4.1 Annotation of Grasp-Type Regions .....	53
3.2.4.2 Annotation of the Task Regions for Task Suitability.....	57
3.2.5 Organization of the Foreground Images and Annotations of Captured Objects into Standard and Irregular Object Sets.....	60
3.3 Creation of the MOMA Synthetic Dataset.....	62
3.3.1 Development of Synthetic Dataset Generation Tool and Procedure for Generating Synthetic Images in COCO JSON and YOLO TXT.....	62
3.3.2 MOMA Synthetic Dataset Summary .....	64
3.4 Object and Affordance Detection Model Architectures.....	69

3.4.1 Mask R-CNN .....	69
3.4.2 YOLACT.....	75
3.4.3 YOLOv5s-Seg.....	80
3.4.4 Multi-Head Mask R-CNN for MAD-Net.....	84
3.5 Experimental Procedure for Training and Evaluating on the MOMA Synthetic Dataset .....	87
3.5.1 Training Loss Criterion for Object and Grasp-Affordance Detection with Instance Segmentation.....	88
3.5.2 Evaluation Metrics .....	91
Chapter 4 Results and Discussion of the Instance Segmentation Models Trained on the MOMA Dataset	93
4.1 Experimental Evaluation of the Instance Segmentation Models during Training .....	93
4.1.1 Training and Validation Loss Curves.....	94
4.2 Experimental Evaluation of the Instance Segmentation Models during Testing .....	98
4.2.1 mAP Evaluation Results for Bounding Box Detection.....	98
4.2.2 mAP Evaluation Results for Binary Image Mask Segmentation .....	101
4.2.3 Inferencing Results for the Test-S Set in the MOMA Dataset.....	104
4.2.4 Inferencing Results for the Test-I Set in the MOMA Dataset.....	110
4.2.5 Summary of the Training, Validation, and the Evaluation Results.....	115
4.3 Final Discussion of the Training, Validation, and Testing Results on the MOMA Dataset .....	118
Chapter 5 Conclusion.....	121
5.1 Future Work and Improvements .....	123
References.....	125

## List of Figures

Figure 1.1: Supination and pronation movements of the forearm and hand. ....	4
Figure 1.2: Single-control system, body-powered cable-operated prosthesis with a hook terminal device. ....	4
Figure 1.3: Examples of terminal devices. ....	5
Figure 1.4: MCPDriver prosthetic finger from Naked Prosthetics. ....	5
Figure 1.5: Example of a myoelectric prosthesis: Open Bionics' Hero Arm. ....	6
Figure 2.1: Examples of prehensile and non-prehensile grasps. ....	11
Figure 2.2: An egocentric frame from the UTG dataset during the collection of video data. ....	15
Figure 2.3: Hand-mounted camera during a grasping task trial from an upper-limb amputee. ....	17
Figure 2.4: Examples of hand-pose estimation predictions from using the MediaPipe model. ....	22
Figure 2.5: Ground truth affordances from IIT-AFF and (b) 3D AffordanceNet. ....	25
Figure 2.6: Comparison of affordance masks as a per-pixel affordance label and independent instances. ....	26
Figure 3.1: Network Architecture of the ResNet-34 FCN model for object affordance detection. ....	33
Figure 3.2: Training losses from the ResNet-34 FCN model for 100 epochs. ....	38
Figure 3.3: Inferencing results on the IIT-AFF dataset for the object affordances. ....	40
Figure 3.4: Sample inferencing results in the IIT-AFF. ....	41
Figure 3.5: Camera setup to capture real single-object RGB images with the OAK-D Camera. ....	43
Figure 3.6: Original images of the captured bottle, and the artificial egocentric view of the object. ....	44
Figure 3.7: All real objects captured for the MOMA dataset, by object class and object set. ....	46
Figure 3.8: All synthetic objects captured for the MOMA dataset, by object class and object set. ....	48
Figure 3.9: Example polygon annotation process on a knife in Roboflow. ....	48
Figure 3.10: Automatic polygon annotation by binary thresholding for synthetic images. ....	51
Figure 3.11: Example of an object with an inaccurate polygon annotation. ....	52
Figure 3.12: Polygon split for the grasp-type annotations from Shapely to LabelMe. ....	54
Figure 3.13: Example where a residual <i>thin rectangle</i> exists after dividing the object image mask. ....	55
Figure 3.14: Point annotation process in LabelMe for the task regions. ....	58
Figure 3.15: Distribution of all objects across all object classes and object sets. ....	61
Figure 3.16: Sample synthetic image generated for the MOMA dataset. ....	62
Figure 3.17: Sample background images of used to generate the MOMA synthetic dataset. ....	67
Figure 3.18: Instance distribution of the number of objects, grasp types, and task regions in MOMA. ....	68
Figure 3.19: Anchor generation example for an input image and feature map from $P_2$ of Mask R-CNN. ....	72
Figure 3.20: Architecture of the Mask R-CNN model. ....	74

Figure 3.21: Architecture of the ResNet-50 FPN Network.....	75
Figure 3.22: Architecture of the YOLACT model.....	78
Figure 3.23: Modified architecture of the ResNet-50 FPN Network for YOLACT. ....	79
Figure 3.24: Architecture of the YOLOv5s-Seg model. ....	82
Figure 3.25: Basic building blocks of the YOLOv5s-Seg model. ....	83
Figure 3.26: Architecture of the mask prototyping network and prediction head of YOLOv5s-Seg. ....	84
Figure 3.27: Architecture of the MAD-Net model.....	85
Figure 3.28: Prediction Head of MAD-Net. ....	86
Figure 4.1: Training and validation plots for detection of instances of objects. ....	95
Figure 4.2: Training and validation plots for detection of instances of grasp types. ....	96
Figure 4.3: Training and validation plots for detection of instances of task regions. ....	97
Figure 4.4: Instance segmentation predictions for the object detection category. ....	105
Figure 4.5: Instance segmentation predictions for the grasp-type detection category. ....	106
Figure 4.6: Instance segmentation predictions for the task-region detection category, <i>carry</i> class.....	107
Figure 4.7: Instance segmentation predictions for the task-region detection category, <i>handover</i> class...	108
Figure 4.8: Instance segmentation predictions for the task-region detection category, <i>use</i> class. ....	109
Figure 4.9: Instance segmentation predictions for the object (bottles, cups, etc.) detection category. ....	111
Figure 4.10: Instance segmentation predictions for the grasp-type detection category. ....	112
Figure 4.11: Instance segmentation predictions for the task-region detection category, <i>carry</i> class.....	113
Figure 4.12: Instance segmentation predictions for the task-region detection category, <i>handover</i> class.	114
Figure 4.13: Instance segmentation predictions for the task-region detection category, <i>use</i> class. ....	115
Figure 4.14: Example where a model incorrectly detected an object but with the correct grasp-types. ....	120

## List of Tables

Table 2.1: Examples of prehensile grasp types Adapted from Feix’s Grasp Taxonomy .....	14
Table 3.1: Camera positioning during real image capturing of the objects .....	44
Table 3.2: Evaluation criteria to classify and categorize each unique object .....	60
Table 3.3: List of specifications used to generate each dataset in the MOMA synthetic dataset .....	65
Table 3.4: Train, validation, and test split of the single-object RGB-A foreground images .....	66
Table 3.5: Mask R-CNN anchor specifications .....	71
Table 3.6: YOLACT anchor specifications .....	76
Table 3.7: YOLOV5s-Seg anchor specifications.....	81
Table 4.1: mAP scores for object bounding box detection (cups, bottles, etc.).....	99
Table 4.2: mAP scores for grasp-type bounding box detection .....	100
Table 4.3: mAP scores for task-region bounding box detection.....	101
Table 4.4: mAP scores for object, binary image mask segmentation (cups, bottles, etc.).....	102
Table 4.5: mAP scores for grasp-type, binary image mask segmentation .....	103
Table 4.6: mAP scores for task-region, binary image mask segmentation.....	104
Table 4.7: Results summary for the detection of object instances (cups, bottles, etc.).....	116
Table 4.8: Results summary for the detection of grasp-type instances.....	117
Table 4.9: Results summary for the detection of task-region instances.....	117



# Chapter 1

## Introduction

Humans are gifted with versatile hands that allow them to complete a plethora of daily tasks, from prehension tasks, such as grasping a mug, to more dexterous tasks, such as typing on a keyboard. The visual system in the human body plays a fundamental role in translating visual data to motor actuation in the hands. Before a grasping task is executed, the brain receives visual feedback from the eyes and perceives the features of all surrounding objects in the environment. A task-oriented grasp is intuitively calculated after evaluating the relationship between the selected object, given task, and context.

In upper-limb amputees, a grasping task is evidently difficult to perform depending on what muscle groups on the upper-limb are lost after an amputation. For example, a transradial amputee (below-elbow) would lose the grasping functionality of a hand, while a transhumeral amputee (above-elbow) would also lose the ability to control the movements of the forearm. Although an upper-limb prosthesis can restore some prehensility (the ability to grasp objects), the selection of available grasps is limited and not task oriented.

### 1.1 Visual Context and Affordances

The visual context perceived in an environment can influence how objects are grasped without altering their state in a scene. For example, a knife is likely used for cutting, if a piece of steak is presented in front of the user; whereas the same knife is likely used for spreading, if both the bread and butter are present instead of the steak. In this example, environmental context changes the task and resulting grasp configuration to hold a knife. In another example, a mug is most likely held at its handle if the containing liquid is too hot, the handle is visible, and/or the mug handle is conveniently oriented in the direction towards the grasping hand; otherwise, the mug is held at its body. Technically, a mug could still be held by the handle if the handle is not visible in a scene. However, the user must have some prior knowledge of the existence of the mug's handle (e.g. if the same mug was visually perceived in a different environment). In the mug example, object context not only influences the resulting grasp configuration, but it also defines the location or task-suitable region to grasp the mug. A grasp configuration, also known as a grasp synergy [1, 2], is a specific hand shape for grasping objects through the collective manipulation of all finger joint angles. If the grasp configurations are generalized to canonical hand shapes, they are referred to as the grasp types [3] (described in Sec. 2.2). Given these examples, visual context is

ultimately semantically reasoned through the opportunities of interactions an environment provides to the perceiver, which are known as the affordances [4, 5]. The environment can represent the setting a perceiver is situated in (such as the kitchen) or the physical objects surrounding the perceiver. In robotic grasping applications, affordances largely describe the object-to-object interactions that humans perceive [6, 7, 5, 8], and the affordances can be categorized as the object affordances [8, 9, 10] or grasp affordances [11, 6, 5, 7, 12]. Object affordances describe the actions an object provides to the user (the perceiver); for example, a cup is drinkable because it affords to contain water. Grasp affordances describes the task locations on the object (also known as a task-suitability region, or a task region for short) [6, 7, 12], and/or the possible grasp configurations to stably (without slipping) grasp an object [5]. For example, the handle of a mug is held with the hook grasp type (as illustrated in Table 2.1 of Sec. 2.2), and it could be associated with the carry, drink, or pour task.

## **1.2 Types of Commercially Available Upper Limb Prostheses and Their Limitations**

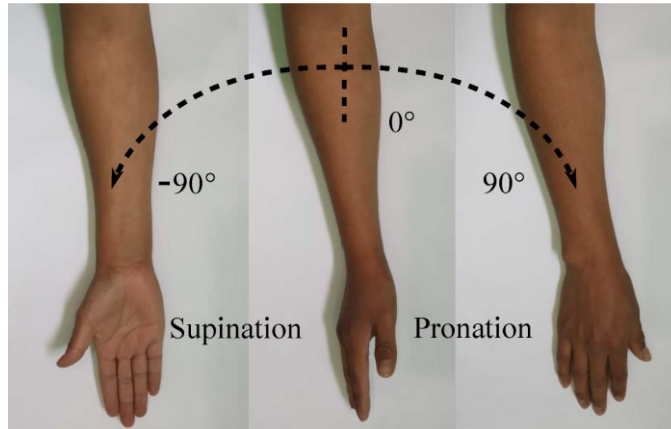
The systematic process of grasping objects may be intuitive to humans. However, developing a vision-based system for robotics and prostheses is complicated due to the number of processes involved. For example, a model is needed for both the trajectory planning of the upper limb, and the grasp selection on an object. Currently, body-powered and myoelectric prostheses are commercially available to restore some of the basic functions of the upper limb, including the ability to grasp objects (prehensility) [13, 14, 15, 16, 17, 18, 19]. However, depending on the severity of the amputation, only certain functions of the upper limb can be restored.

### **1.2.1 Body-powered Prostheses**

Early functional prosthetic arms are body-powered and cable-operated, as seen in Fig. 1.2. An amputee wears the prosthesis on their residual limb with an accompanying shoulder harness on their shoulders. A single tension control cable connects the shoulder harness to the prosthetic arm, terminal device, or both [13]. A terminal device (Fig. 1.3) is a prosthetic hand or hook that fits onto the wrist of a prosthetic arm [13]. Other task-specific tools could also be attached to the terminal device, such as a fork for eating. The overall control strategy of the prosthesis differs depending on the existence of the elbow on the residual limb. There are two types of shoulder harnesses for a body-powered cable-operated prosthesis: a single-control system for transradial (below-elbow) amputees, and a dual-control system for transhumeral (above-elbow) amputees [13]. In all cases, the terminal device will correspondingly release or grasp

objects when the residual limb is extended or relaxed. The opposite shoulder can also control the terminal device by flexing or relaxing. For a dual-control system, the control cables are also fixed onto the upper arm and forearm of the prosthesis to produce flexion and extension movements of the forearm. In both cases, the movements of the arm and terminal device are coupled. Body-powered prostheses also have limited control of the other movements of the arm, such as the supination and pronation (Fig. 1.1) of the forearm, are either naturally adjusted by the residual limb or the opposite hand. In all levels of amputation beyond a wrist disarticulation (the amputation at the wrist of an upper-limb [20]), the amputee cannot control the orientation of the wrist with a body-powered prosthesis [13].

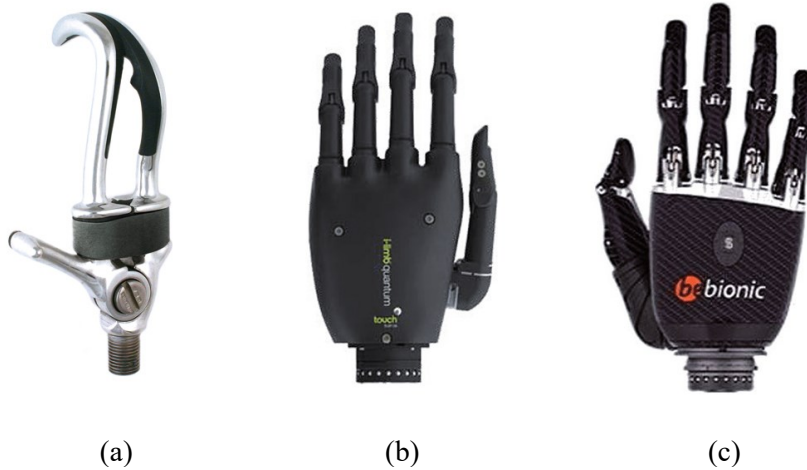
The terminal device is operable in all levels of an upper-limb amputation with limited functionality. Only a single prehensile grasp type can be achieved since the opening and closing mechanism couples the manipulation of all finger joints to a single degree-of-freedom (DOF). Prehensile grasps include all grasps that a hand can hold an object stably without slippage [21, 22]. The ability to control the tension control cables to operate the terminal device also becomes increasingly difficult as the severity of an upper-limb amputation increases. If an amputee has a partial-hand amputation, they still have the option to restore individual finger control for more varying grasp types. The traditional cable-operated prosthesis is replaced with a wearable prosthetic finger (Fig. 1.4) for each of the missing digits, as long as the proximal phalange (the bone above the knuckle) is preserved to fit the prosthetic finger [15]. The prosthetic fingers are composed of mechanical linkages, where finger flexion is achieved when the residual finger pulls onto the attached suspension ring [15]. For upper-limb amputees beyond a partial-hand amputation, myoelectric prostheses could replace the body-powered prostheses for more independent control of the arm and hand. Myoelectric prostheses also provide a larger selection of unique grasp types for a grasping task.



**Figure 1.1:** Supination and pronation movements of the forearm and hand, adapted from [23].



**Figure 1.2:** Parts of a single-control system, body-powered cable-operated prosthesis with a hook terminal device. The terminal device is controlled through a control cable, that is connected to the shoulder harness. Adapted from [24].



**Figure 1.3:** Examples of terminal devices. (a) for a body-powered prosthesis, (b) and (c) for a myoelectric prosthesis. (a) Hosmer 88-x Hook adapted from [14], (b) Össur i-limb Quantum Bionic Hand (adapted from [24]), and (c) Ottobock BeBionic Hand (adapted from [24]).



**Figure 1.4:** MCPDriver prosthetic finger from Naked Prosthetics. Adapted from [15].

### 1.2.2 Myoelectric Prostheses

A myoelectric prosthetic arm (Fig. 1.5) enables an amputee to control the arm and terminal device by recognizing muscle activity patterns with surface electromyograms (sEMG). Electrodes are placed on the muscle groups of the residual limb to produce multi-channel sEMG readings [25, 26, 2, 27]. Most movements of an arm are restored with a myoelectric prosthesis, except for wrist rotation. However, a motorized wrist joint can achieve active wrist rotation, by pressing a button with the opposite hand (Fig. 1.4) [17]. Myoelectric prostheses can offer up to eight different pre-programmed prehensile grasp types at a time [17, 18, 16, 19].

Although myoelectric prostheses solve most of the limitations of body-powered cable-operated prostheses, the number of available grasp types are still limited to a small subset of all possible grasp types. Manual control from the intact hand is still required to cycle through the alternative grasp types. The simultaneous sEMG pattern recognition of both the arm movements and grasp types may also overwhelm the user, due to the complexity of the control strategy [28]. A precise muscle movement from the amputee is required to trigger a specific sEMG pattern for an intended action [25, 27]. Recent prosthetic devices overcome the problem of the coupling control strategy in pure myoelectric prostheses, by detecting the grasp types with computer vision first, before action intent and arm movements are determined through sEMG pattern recognition [25, 26, 27]. With computer vision, a more suitable grasp type can be selected for a corresponding grasping task [25, 29].



**Figure 1.5:** Example of a myoelectric prosthesis: Open Bionics' Hero Arm. The circular button on the back of the terminal device is the program switch to cycle through an alternative set of grasp types. Adapted from [24].

### **1.3 Vision-based Prosthetic Grasping and Related Vision-based Methods in Robotic Grasping**

Vision-based prostheses use a red-green-blue (RGB) camera to capture images or video scenes of the objects in the environment [25, 27, 30]. The visual data from the camera is then used to train a deep-learning model to predict grasp types for the detected objects. The deep-learning model used is typically a deep convolutional neural network (CNN), where a grasp type is classified for each object detected in a scene. The camera can be mounted on the back of the hand [25, 27, 30] or worn on the head [31, 32, 30]. If the camera is head mounted, multi-object scenes can be detected while surveying the actions of other

hands (such as the user's intact hand) [31, 32]. Most prosthetic grasping methods opt for the hand-mounted camera to detect single objects, while eliminating the need for additional object-and-grasp-selection models [25, 26, 29, 27, 30]. However, single-object scenes lose the visual context about other objects and the environmental setting, since only the features of the selected object influence the final grasp-type prediction in current vision-based prosthetic grasping methods [25, 26, 29].

Currently, most of the issues involving the movements of the prosthetic arm are already solved with myoelectric solutions [25, 26, 17, 16, 19, 2]. However, the available selection of task-oriented grasps for terminal devices is still lacking in myoelectric and vision-based prostheses. For myoelectric prostheses, the aforementioned problem can be attributed to the fixed pre-programmed grasp types [17, 18, 16, 19] and the inability to utilize object features to predict a grasp (e.g. the object's overall shape). With the introduction of vision-based prostheses, the object's class predetermines the resulting grasp-type predictions. The class of an object is the label or name used to denominate similar objects. The main issue with current vision-based methods is the assumption that all objects have one possible grasp type to grasp the object, no matter the object's environmental context and the intended task a prosthetic user wants to perform. The same class of objects are grasped in the same manner, despite the variances in the object's shape, pose, parts, and functions (the affordances). No alternative grasp types are offered to the prosthetic hand even if the predicted grasp type fails to grasp the selected object. The object pose describes the orientation and position of the object in a scene. Current vision-based prostheses also prioritize the selection of stable grasps over task-oriented grasps [25, 26, 29]. While visual data can be leveraged for grasp selection, there are currently only a limited number of vision-based methods implemented in prosthetic grasping [27, 25].

Recalling Sec. 1.1, robotic grasping applications that leverage affordance detection for the selection of task-oriented grasps can be adapted to improve the vision system in prosthetic devices. In robotic grasping, affordances relate to visual context of the whole scene. Other objects in the environment are interpreted by the deep-learning model as well. Affordances enable deep-learning models to perceive the different object parts and characterize each object part with a different task. For a given RGB image, affordances are typically detected by a deep-learning CNN segmentation model as image masks, bounding boxes, or both [8, 7]. Therefore, this thesis will focus on the development of a visualization model for vision-based prostheses using affordance detection methods adopted from robotic-grasping applications.

## 1.4 Research Objectives

The main goal of this thesis is to develop a deep-learning model that can simultaneously detect all objects and their grasp affordances in RGB multi-object scenes for application to prosthetic grasping. Ultimately, the deep-learning model is a deep CNN segmentation model that will find the semantic relationship between objects and grasp affordances. To accomplish this objective, a list of specific objectives is outlined below:

- a) Create a dataset that provides image mask annotations of objects and their grasp affordances (as grasp-type and task regions) for every RGB image.
- b) Train and evaluate the performance of several existing deep CNN segmentation models for the separate detection of object and grasp-affordances, using the dataset created in objective (a). A segmentation model is therefore trained and evaluated three times on the dataset, once per detection category (objects, grasp types, task regions).
- c) Develop a new multi-tasking deep-learning model that jointly trains the detection of objects and their grasp affordances on RGB images.
- d) Evaluate the performance of the new model in (c) for jointly detecting objects, and grasp-affordances. Determine the feasibility of the joint detection of objects and their grasp affordances in comparison to the trained models in (b).

## 1.5 Research Contributions

The main research contributions in this thesis are the creations of a new synthetic dataset and a deep CNN segmentation model that detects multiple grasp-affordances (as grasp-type and task regions) and objects in a scene, in application to vision-based prosthetic grasping. The specifics of each contribution are outlined below:

- a) The Multi-Object Multi-Grasp-Affordance (MOMA) synthetic dataset that consists of 20K RGB images of multi-object scenes in varying object poses was newly developed. Each RGB image contains annotations for the detections of objects and their grasp affordances. The grasp affordances are further subdivided into two categories of annotations: task-oriented grasp-types and task regions. The details about the creation of the MOMA dataset are given in Sec. 3 of this thesis.
- b) A Multi-Affordance Detection Network (MAD-Net) model was newly developed to accommodate the creation of the MOMA dataset. MAD-Net is a variation of the traditional object



instance segmentation model, Mask R-CNN [33] that predicts instances of objects, and their grasp affordances as task-oriented grasp-types and task-suitable regions. Additional details about instance segmentation are given in Sec. 2.4.1, and the development of the MAD-Net model in Sec. 3.4.4 of this thesis.

## **1.6 Thesis Outline**

In this chapter, the motivation and rationale for the thesis objectives were discussed. Chapter 2 discusses the definitions of grasps, affordances, and task-oriented grasping in applications to prosthetic and robotic grasping. Chapter 3 summarizes the reasoning and steps leading to the creation of the MOMA synthetic dataset. In addition, Chapter 3 discusses the models that were trained and evaluated on the MOMA dataset for the detection of objects, and their grasp affordances (as grasp-type and task regions), as well as the evaluation methods used on all the models. Chapter 4 provides the results and discussion of the experimental evaluation of all models described in Chapter 3. Chapter 5 concludes the findings of this thesis in relation to the research objectives in Sec. 1.4.

## **Chapter 2**

### **Background and Literature Review**

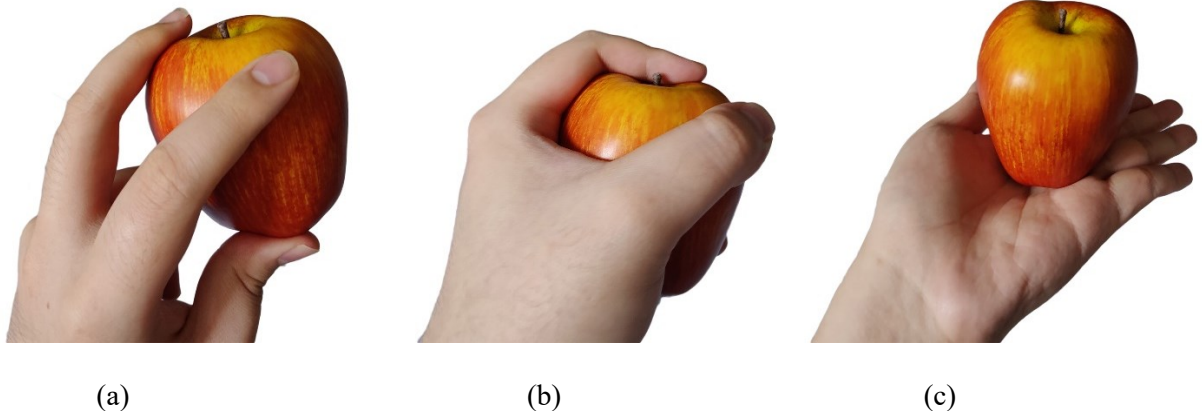
In this chapter, current vision-based prosthetic and robotic grasping methods are reviewed. Sec. 2.1 provides details of grasp definition and object manipulation by a hand. Sec. 2.2 outlines how grasps can be categorized into a grasp taxonomy of unique grasp types and utilized for grasp-type classification in videos of activities of daily living. Sec. 2.3 explains how grasps are estimated and represented in current prosthetic and robotic grasping applications. Sec. 2.4 and Sec. 2.5 describe the implications of choosing stability over task-suitability for a task, and how a grasp can be selected by identifying affordance regions. Sec. 2.6 provides a synopsis of current prosthetic and robotic grasping methods, as well as the rationale for the thesis objectives that was outlined in Sec. 1.4.

#### **2.1 Definition of a Grasp**

Humans can interact with objects by using one hand through prehensile (grasping objects without slipping [21, 22]) and non-prehensile manipulation, or with two hands through bimanual manipulation.

Throughout this thesis, a hand refers to anything that enables an object to be picked up, unless otherwise specified: a human hand, a robotic hand or gripper, and a terminal device for a prosthesis. All prehensile object manipulations are prehensile grasps since the object and the hand can leave a surface as a single conjoined entity [22, 21]. Conversely, all non-prehensile object manipulations are not grasps, although they are classified as a non-prehensile “grasp” in a grasp taxonomy (discussed in Sec. 2.2) [21, 22, 32]. As seen in Fig. 2.1a, an apple is grasped if the apple is enveloped. An apple is not grasped but merely supported by the hand with non-prehensile manipulation (non-prehensile grasp) in Fig. 2.1c. Examples of prehensile grasps are shown in Figs. 2.1a and 2.1b, and non-prehensile manipulation in Fig. 2.1c.

Bimanual object manipulation (by both hands) can also be classified as prehensile or non-prehensile. In this thesis, only one-handed prehensile grasps will be analyzed.



**Figure 2.1:** Examples of (a, b) prehensile and (c) non-prehensile grasps. (a) also represents a force-closure grasp as there are three defined contact points on the apple, whereas (b) is a form-closure grasp since the hand completely wraps around the apple to form infinite contact points.

### 2.1.1 Prehensile and Non-Prehensile Grasps

Prehensile grasps are the most analyzed subset of all grasps involving hands interacting with an object [31, 32]. Approximately 60% of all grasps by upper-limb amputees are prehensile, where less than 10% of all prehensile grasps are performed on a prosthesis [31]. As mentioned in Sec. 1.2, upper limb amputees find it difficult to operate a body-powered and a myoelectric prosthesis due to the lack of functionality and the complex control strategy [28]. Although myoelectric prostheses provide more unique grasp types and remove the need to operate a prosthesis primarily by muscles, the prosthesis usage for prehensile grasps between the two types of prostheses remains unchanged [31]. There is thus a need to improve the prehensile grasping capabilities on a prosthesis. Therefore, only prehensile grasps will be analyzed in this thesis.

In robotic applications, a prehensile grasp is defined as the ability of a hand to achieve force closure by maintaining contact on a held object [21, 32]. Although this thesis research will not focus on the force analysis of prehensile grasps, the robotics definition of a prehensile grasp is important to understand how prehensile grasps are classified and organized in a grasp taxonomy (discussed in Sec. 2.2). Force closure (Fig. 2.1a) defines how an object is stably constrained to a hand without internal movements (slippage) by balancing external wrenches at the contacts [21, 34]. A wrench at a contact point is a vector that contains both the resultant force vector from the induced friction cone, and the generated torque between the grasped object's center of mass and the contact point. The friction cone at a contact point is modelled as a

conical shape to represent the possible directions that the friction force is exerted on an object [35, 34]. For a grasp to be stable, the resultant friction force from the friction cone must be in equilibrium with the counteracting normal and finger contact forces [35]. If the supposed grasp also resists all external forces and moments in any direction, then the grasp is said to have form closure (Fig. 2.1b) [21]; though, achieving only force closure is enough to define a prehensile grasp. All grasps that allow the environment (such as gravity) to affect the movements of an object are classified as non-prehensile grasps [22]. An example of a non-prehensile grasp is illustrated in Fig. 2.1c, where the object is supported by the palm of the hand without other opposing finger forces. Other examples of a non-prehensile grasp include the manipulation of objects by pushing or pulling against a surface.

In the case of a form closure grasp, there could be an infinite number of contact points between the hand and object. If there are at least two opposing sets of fingers that are in equilibrium with an object, then the grasp is also a prehensile grasp [22]. A set of fingers that apply a similar direction of force on an object is called a virtual finger, where the palm of a hand could also be a virtual finger by itself; each virtual finger produces a resultant finger contact force with an object [22]. For example, in Fig. 2.1b, there are three virtual fingers: from the thumb, from the palm of the hand, and from the index finger to the pinky. Each set of virtual fingers act in opposition to keep the apple stably in the hand.

The classification of force closure and form closure grasps is useful to further subdivide all prehensile grasps based on how the finger forces of the hand are applied on the surface of an object. In Fig. 2.1a, the fingertips of the hand maintain force closure with an apple, where the apple can easily be further manipulated by the hand in the case of a dexterous task. However, in Fig. 2.1b, the apple is grasped in a way that is fully enveloped by the hand, where further manipulation of the apple would be difficult. The grasp configurations on the object in Figs. 2.1a and 2.1b can therefore be categorized as a precision grasp or a power grasp, respectively. If a grasp has characteristics of a precision and power grasp, then the grasp is categorized as an intermediate grasp [22].


## **2.2 Grasp Taxonomy**

All hand-object interactions – regardless of whether the hand action is prehensile or non-prehensile – can be generalized to a set of hand poses of unique grasp types categorized in a grasp taxonomy [36, 22, 37, 38, 21, 3, 31, 32]. Each hand pose in a grasp taxonomy can be extracted for a default set of joint parameters using hand-pose estimation models [39, 40, 41]. These parameters correspond to each DOF of the fingers joints and wrist joint to form a specific grasp configuration. When hand poses are used to

estimate a specific joint configuration to grasp an object, they are known as the grasp pose or grasp configuration [6]. Otherwise, all generalized pose estimates of a hand are described as hand poses. Most grasp taxonomies only analyze prehensile grasps since most grasping applications require the capability to estimate stable grasp configurations for pick-and-place tasks [22, 37, 38, 21]. Prehensile grasp types are classified based on the quality of the grasps, whether the intended grasp emphasizes more on stability or dexterity; therefore, each grasp would be classified as a power, precision, or lateral (intermediate) grasp [31, 32, 22] (Table 2.1).

The creation of grasp taxonomies is rather ambiguous. The vague definition of unique grasp types has led to multiple interpretations of grasp taxonomies with varying cardinalities [36, 22, 37, 38, 21]. For instance, Cutkosky's grasp taxonomy [21] identifies 15 unique prehensile grasps defined from object geometry, which includes the object's overall shape, and dimensions [21]. In contrast, the GRASP taxonomy [22] contains 33 unique prehensile grasps when object features, positioning of the thumb, and configurations of the virtual finger forces are considered. Alternatively, there are 17 unique prehensile grasps [22] in a grasp taxonomy if the shape of the object is not considered. Some of these identified grasp types also share similar characteristics among the existing grasp types in the taxonomy. When grasp taxonomies are applied in deep learning models, these redundant grasps will increase the inaccuracies in classification and in the ground truth annotations of datasets. For example, the GRASP taxonomy [22] indicates that there is a 69.7% chance for the misclassification of tripod grasps with lateral grasps, even though human experts manually annotated each video sequence. Overall, grasp taxonomies provide a simple basis to describe the most common grasp types and their associations with object features.

**Table 2.1:** Examples of prehensile grasp types adapted from Feix’s Grasp Taxonomy [22] and [42]. Each grasp type is organized into power, precision, and lateral subcategories.

Power	Precision	Lateral (Intermediate)
<p data-bbox="293 401 500 432">Cylindrical Wrap</p> 	<p data-bbox="721 401 898 432">Precision Disk</p> 	<p data-bbox="1187 401 1263 432">Distal</p> 
<p data-bbox="293 690 500 722">Adducted Thumb</p> 	<p data-bbox="699 690 922 722">Prismatic 2 Finger</p> 	<p data-bbox="1138 690 1312 722">Lateral Tripod</p> 
<p data-bbox="277 968 516 999">Hook (Fixed Hook)</p> 	<p data-bbox="699 968 922 999">Prismatic 4 Finger</p> 	<p data-bbox="1170 968 1263 999">Ventral</p> 
<p data-bbox="293 1241 500 1272">Index Extension</p> 	<p data-bbox="764 1241 857 1272">Tripod</p> 	
<p data-bbox="326 1541 467 1572">Light Tool</p> 	<p data-bbox="699 1541 922 1572">Parallel Extension</p> 	

### 2.2.1 Grasp Type Recognition in Human Activity Datasets

In recent grasp analysis applications, wearable head-mounted cameras have been increasingly used to capture images and videos in a first-person view [31, 32, 36, 37, 38]. As seen in Fig. 2.2, a video captured in first-person view (FPV), is also known as an egocentric video. Egocentric videos in grasping applications capture timelapses of human activities with evolving hand poses. The FPV of egocentric videos maintains a similar viewpoint as if the camera were fixed onto a rigid frame: overlooking a scene, but in motion. Currently, there is no known method that applies egocentric video to prosthetic grasping models. However, there are some experiments conducted to analyze the grasp distributions among individuals with an upper limb unilateral amputation [31, 32]. With egocentric video, grasp types can be automatically detected by using machine-learning models [36, 37] to learn hand poses, unlike conventional means of manual annotations [31, 32, 22] on videos. Specifically, egocentric videos fully capture the perspective of human grasping which allows an accurate representation of the many grasp types utilized in daily human activities.



**Figure 2.2:** An egocentric frame from the UTG dataset [38, 37] during the collection of video data. The user performs a grasping task that is recorded by a head-mounted camera.

The UTG Dataset [37, 38] is an example of a FPV human-activity dataset used to automate prehensile grasp-type detection. It contains egocentric videos of five participants grasping unique objects on a table in an office setting, where each object is grasped multiple times using one of the 17 unique grasp types from the GRASP taxonomy [22]. Despite the variations in the available objects, the general finger poses, number of contact points, and physical appearance of each unique grasp type remains relatively constant.

However, since the trajectory to approach an object and the object shape are different for each instance, the fingers and wrist poses need to adjust accordingly to comfortably wrap around the object. Therefore, the UTG dataset has changing wrist poses for each grasp action and multiple objects that require the same grasp types. The UTG Dataset [37, 38] also includes a machine-learning model to automate prehensile grasp analysis. Each hand region is first pre-processed with a feature descriptor to detect low-level features by computing the local intensity gradients in an image. The sharp contrast between neighbouring pixels is a common indicator of an object edge. A Histogram of Oriented Gradients (HOGs) and a Scale-invariant Feature Transform (SIFT) are examples of feature descriptors utilizing local pixel intensities for edge detection. These encoded feature descriptors are then trained on a classifier to identify the grasp types, such as a Support Vector Machine (SVM) model that can handle data with high dimensionality. For automatic taxonomy generation, the identified grasp types are grouped together based on the correlations of hand postures with hierarchical clustering methods [36, 37].

### **2.3 Grasp Estimation and Representation**

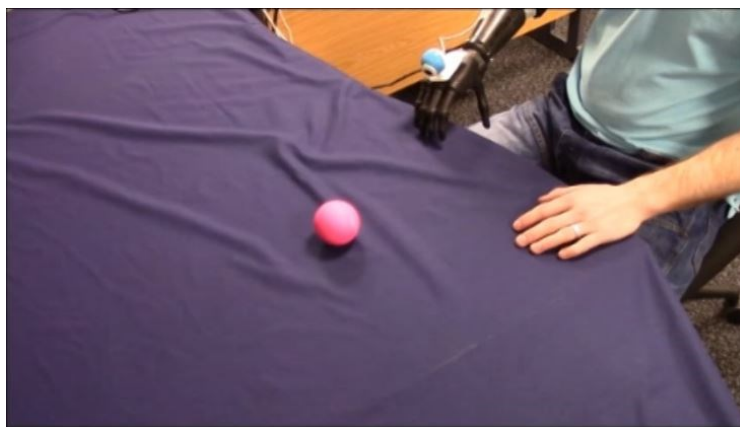
Computer vision is commonly incorporated into robotic systems for visual perception of the environment. Current developments in robotic grasping implement vision systems to directly identify potential grasp poses from object features [43, 44, 45, 46, 6, 10, 7, 12], observed hand poses for grasp analysis [36, 37, 47, 38, 40, 48, 41, 3], or a combination of hand-object features [11, 49, 50, 37, 40, 5, 3]. Traditional prosthetic grasping solutions have been primarily focused on pattern recognition of myoelectric signals through sEMGs on the forearm [25, 26, 2, 51, 52]. Recent advancements predict 3D hand poses from sEMG signal patterns; however, these methods have yet to be implemented for grasping applications [51, 52]. Robotic grasping models can detect grasps for multi-object cluttered scenes, while prosthetic grasp models are limited to single-object scenes. In recent prosthetic grasping methods, there is an emerging interest to combine myoelectric signals with image data to predict grasps [25, 26, 51, 52].

Image recognition is increasingly applied to existing myoelectric prostheses, eliminating the need for decoding noisy sEMG signal patterns to estimate grasps [25, 26, 43]. However, sEMG signals are still necessary for predicting grasp intent and motion trajectories for the arm to approach an object. During the experimental evaluation of myoelectric and myoelectric-vision models, the tests are either conducted with simulated prostheses on abled individuals [26], or amputees with existing prosthetic devices [25]. When robotic systems are utilized in simulating grasps for a prosthesis, a robotic hand with varying dexterities is



fixated on a robotic manipulator. In this case, the robot arm is controlled by a human operator via manual control [43], or teleoperation [2].

The choice of placement of the cameras is crucial to achieve the highest possible accuracy in vision models, since cameras suffer from occlusions and distorted frames, resulting from an obstructed camera view and high-speed camera motion, respectively. Placing the camera on the hand (Fig. 2.3) may contribute to consistent detection errors in a single trial, leading to significantly longer runtimes to complete a task [25]. Since the hand moves jointly with the camera, the overall movement may cause streaking in the image, leading to poorly defined edges representing an object. During the execution of a task, the camera is unable to capture the entire object as it approaches too close to an object. Although an image of the object can be captured before the hand approaches the object, the hand may not be in a position to orient the camera to see the object at all. For example, the hand is in a resting position on an individual's side where the camera faces towards the ground. However, the selected object is left undetected if it is placed on a table above the hand and camera. Pre-grasps can also obstruct the view of the camera, which will also eliminate the possibility of detecting an object beyond the occlusion. A pre-grasp is a premature grasp configuration used to prepare a hand, moments before an object is grasped. In simulated vision-based myoelectric prostheses, the camera is fixed onto a tripod to face and overlook both the arm with a gripper and surrounding objects [26, 43]. In application to prosthetic grasping, however, this method would be impractical to implement on a prosthetic hand since the user is situated in a changing environment. Therefore, recent robotic applications analyze grasps in an egocentric view to imitate grasps from a human's perspective [53, 42, 49, 50, 47, 3].



**Figure 2.3:** Hand-mounted camera during a grasping task trial from an upper-limb amputee. Adapted from [25].

### 2.3.1 Grasp-Type Classification and Detection for Grasp-Pose Estimation

Recent vision-based grasp estimation models leverage edge detection and spatial invariance properties of deep CNNs, for both robotics [53, 49, 50] and prosthetic-grasping applications [25, 26, 43]. To estimate a suitable grasp pose using grasp types, a CNN with multiple convolutional and pooling layers is first deployed as feature extractors to learn the image embeddings of a given RGB scene. These image embeddings are then passed through a fully connected layer with a sigmoid activation function to predict the most probable grasp type for the detected object shapes [25, 26, 42]. As mentioned in Sec. 2.2.1, other feature extractors can replace a CNN network. However, CNNs recognize a hierarchical representation of features, while HOGs and SIFTs only describes low-level features (such as edges) based on pixel intensities. Since each convolutional layer contains trainable weights, CNNs are adaptable to changes between every training example and generalize better to unseen examples.

The aforementioned CNN structure only works well if the predictions are made for single object scenes. Since object selection is not necessary, a single grasp type is directly predicted on the one object. In application to prosthesis users, however, most settings take place in cluttered scenes with multiple objects in changing scenes. Multi-class classification can still be directly applied to the entire image frame in a single iteration. Bounding-box regression models for object detection, such as YOLO [54], SSD [55], and Faster R-CNN [56] localize image patches of all identifiable objects in a scene. Each of these image patches are then classified in a single pass for its grasp types in a CNN classifier [5, 8, 29, 47]. However, current methods still only predict a single grasp type for each object, in both grasp type classification [25] and grasp-type detection [29, 47] models. Grasp-type classification has also been done where multiple grasp types are assigned to each unique object [27]. However, the user still needs to proactively reject each unsuitable grasp type for the grasping task, and an increase in runtime to complete the grasping task can therefore be expected [27]. The main problem with current vision-based methods for grasp type selection is that the grasp types are not selected for the suitable object parts, but rather, the objects as a whole.

The same deep CNN models used for images are also applicable to grasp type analysis in videos [42, 3] and ultimately for physical grasp execution [53, 49, 50]. RGB videos adds an extra temporal dimensionality to conventional RGB image datasets, which allows for new features that describe the time evolution of grasp poses and the trajectory paths to approach an object [42, 40, 41]. In conjunction with a CNN, a mean-shift object-tracking algorithm can continuously track a localized hand throughout a video to understand how grasp poses change during tasks [42]. By implementing localization algorithms for

both hands and objects in a video, it is possible to distinguish grasp poses from casual gestures and estimate action intent [42].

An RGB image can be plagued with colour noise, and the indistinguishable colours between the foreground and background negatively impact how well a CNN model can identify objects based on their shape. To solve this problem, in recent research, CNN models have been trained with a multimodal dataset to extract and learn features from multiple perspectives [26, 29]. For example, using just RGB images results in a classification accuracy of 81.16%; however, by converting RGB images to grayscale images and complementing the result with depth images, the classification accuracy improved by 12.75% [26]. The same accuracy improvement can also be seen in a grasp-type detection model when multimodal data of RGB and depth images are used [29]. Depth images provide a pseudo image segmentation of all objects in a scene, resulting in more defined edges on each object. When the RGB images are converted to grayscale images, the colour features are removed, while providing some information about the overall 3D geometry and texture on the object. Therefore, by leveraging both modalities of data, the classification and detection accuracy can significantly improve the performance of grasp-type detection and grasp-pose estimation models [26, 29].

### **2.3.2 Object-Pose Estimation in Grasp-Pose Estimation**

Grasp poses are largely described by detected object features, including an object's shape and object's class. As discussed in Sec. 2.3.1, deep CNNs for grasp-type classification intuitively select grasp types that closely match the selected object and its detected shape. All objects can also be described by their pose. An object in a 3D space is vectorized as a 6-DOF pose (commonly referred as a 6D pose), which contains information about its position and orientation within a scene. For every instance the same object appears in a scene, its pose relative to the camera changes if the camera position is not fixed, or the objects are relocated. Therefore, raw grasp-type poses classified in a CNN need to be refined to ensure contact is made on the object. Grasp-type estimation is still necessary as it provides a simpler solution to reduce the number of possible grasp poses on an object. Since all prosthetic and robotic hands have varying DOFs, the same grasp pose may not be applicable to all hands. With grasp-type estimation, a general grasp pose can be determined and adapted by all hands with minimal corrections needed to establish contact with an object. In this thesis research, only grasp types are used to determine how an object should be grasped. However, grasp-pose estimation methods are still discussed in this section to provide some details of how grasp-type estimation can be applied or replaced in grasp-pose estimation

models. Most grasp-pose estimation methods require the initial estimation of the object pose [49, 50, 37, 40, 5, 3].

In the GanHand model [5], the 6D pose of all detectable objects is computed from a single RGB image, while simultaneously learning the 3D geometries of the detected objects. The GanHand model is a Generative Adversarial Network (GAN) that can receive input RGB images with one or more objects in a scene, and generate multiple grasp poses on a single object. In multi-object scenes, the object pose is directly estimated from the RGB image, and the model additionally produces the 3D shape of the object. In contrast, the objects in single-object scenes are first reconstructed as 3D point clouds with AtlasNet [57]. The AtlasNet model is trained on the ObMan dataset to choose a synthetic 3D object model that closely resembles the shape of the detected object. The resulting 3D object shapes are then projected onto a 2D segmented mask and concatenated with the original RGB image to predict the grasp type; only one grasp type is predicted for each object, despite the multiple grasp poses for the same object. The grasp type is then converted to a 51-DOF MANO [58] representation to produce realistic hand meshes with human-like hand configurations. Finally, the 51-DOF MANO representation of the grasp type is optimized in the hand refinement network. In the hand refinement stage of the GanHand model [5], the selected grasp types are converted to grasp poses. Each finger on the 51-DOF MANO representation is moved to establish contact with the target object, before the arc lengths formed between the distal, proximal, and knuckle joints of each finger are minimized. Although the GanHand model [5] can predict realistic grasp poses, the selected grasp poses are not necessarily task oriented. In addition, the GanHand model [5] may not be feasible for prosthetic grasping and real-time applications since the grasp poses are estimated one at a time for a given multi-object scene.

There are other alternative methods to estimate object-based grasp poses without the need to classify grasp types [43, 44, 45, 50, 49, 6]. Similar to the GanHand model [5] for single-object scenes, a deep CNN can first be trained from scenes captured in a simulation environment (such as Gazebo [59]) before performing the grasping tasks on real scenes [43]. In this case, the simulation environment captures both an RGB and depth image of the simulated scene, and an initial grasp pose on the target object is also generated with GraspIt! [60]; GraspIt! [60] is a grasp planner that allows the user to generate stable grasp poses on a chosen gripper. Instead of directly feeding the generated grasp pose to a deep learning network, a suitable palm patch is extracted from the depth map and the vectorized palm pose from the generated grasp plan. The original grasp pose produced from GraspIt! [60] serves as a ground truth annotation to evaluate the deep learning model [43]. A final grasp pose is then computed in a Grasping

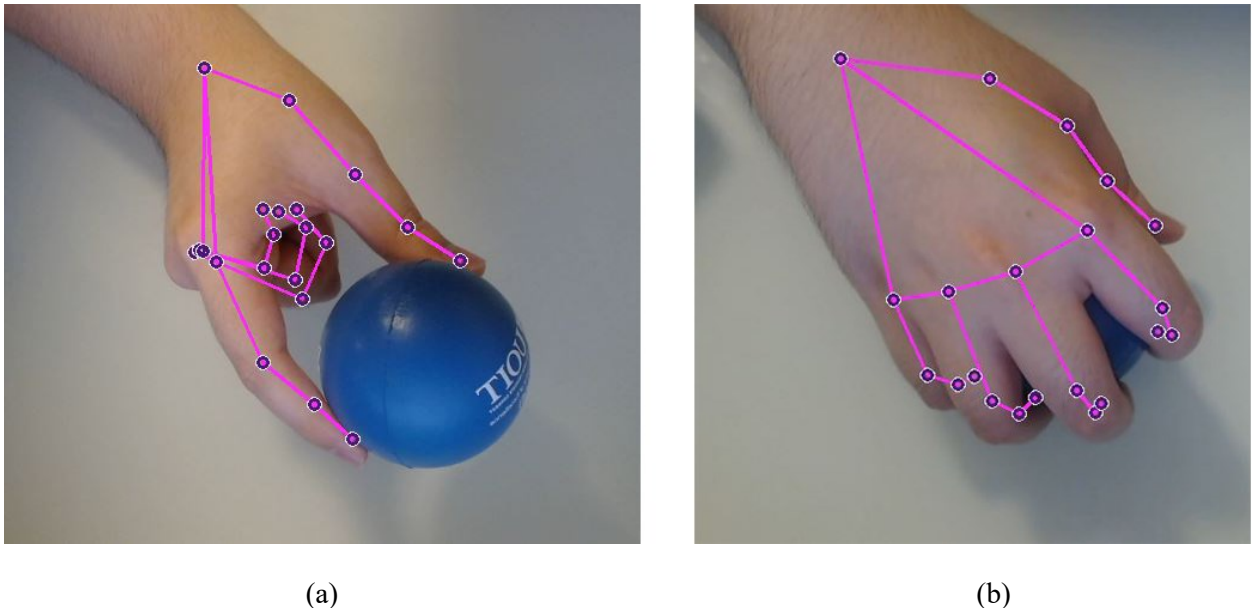
Posture Prediction Network (GPPN), which predicts the overall alignment of a gripper’s palm on the target object and the corresponding finger poses. The intuition behind this method is that the alignment of the palm is inherently important to find the specific grasping posture for the task, even if the palm of the hand is not in contact with the object (e.g. pinch grasp); the finger poses can always be fine-tuned to produce a stable grasp pose by maximizing the grasp quality index [43, 34, 35]. Objects can additionally be further dissected into their primitive shapes (cubes for example) and trained independently on a deep CNN network and generalize to more complex shaped objects (everyday objects such as a mug) [60, 45].

### **2.3.3 Applications of Hand-Pose Estimators in Grasp-Pose Estimation**

As discussed, in Sec. 2.3.2, grasp estimation tasks are commonly solved by first identifying a target object, followed by the estimation of a suitable grasp pose from the extracted object’s features, which includes its shape, class name, and pose. Recalling Sec. 2.2, all general pose estimates of a hand are the hand poses. A hand pose cannot be directly used as a grasp pose, since only the detected object features, such as an object’s shape, can accurately allow contact between the object and a hand. However, general hand-pose estimations are still useful to understand common hand structures for applications in grasp-type analysis [36, 37, 47, 40, 48, 41, 38, 3] as a general hand pose predetermines a general pre-grasp hand shape. Recent methods combine hand-object poses to learn realistic human hand-grasp poses in real scenes [49, 50, 37, 40, 5, 3], eliminating the need for training models on synthetic datasets with software-generated artificial grasp poses [44, 5].

Human hand poses can be estimated in several ways: from a capacitive soft sensing stretch glove like the CyberGlove [61, 2, 62], sEMG pattern recognition [51, 52], 3D point cloud data of objects [49, 5], or vision-based keypoint detection models to localize and structure the human hand as a 21-jointed skeletal kinematic chain, or a 21-jointed grasp configuration (Fig. 2.4) [49, 50, 39, 40, 41, 52, 51]. All of these methods have also been widely applied for real-time applications to continuously track hand poses. Vision-based techniques can be adapted to estimate hand poses as the locations of a 21-joint hand configuration (Fig. 2.4) directly from image and video data [49, 50, 39, 40, 41]. In application to prosthetic grasping, current deep learning models, such as NeuroPose [52], associate detected sEMG patterns on the hand with a 3D hand pose, provided that there are additional annotations of captured 3D hand poses from hand-tracking devices or software. Current hand-pose-estimation models, such as MediaPipe [39], are modelled as a classification problem to identify the positions of each joint as a keypoint on the detected hand. Since an image is projected to a flat 2D plane, recent studies use CNNs to

recover the missing third dimension from depth data, to fully estimate the 3D coordinates of each joint in the hand pose [49, 50]. Alternatively, there are known 3D hand pose datasets that provide a complete set of 3D hand poses with corresponding object poses in image and video scenes. These hand poses are acquired by attaching multiple magnetic sensors on the grasping hand during the execution of a task [40, 41]. The First-Person Hand Action Benchmark (F-PHAB) dataset [40] is an example of a 3D hand-object pose dataset that also allows the user to train an action-recognition model in egocentric videos. In applications to robotic grasping, the 6D pose of a hand can either be directly incorporated into a 21-joint grasp configuration [41] or estimated separately [49] as a regression problem with CNN models.



**Figure 2.4:** Examples of hand-pose estimation predictions from a hand-object interaction using the MediaPipe [39] model. Each hand pose is a grasp pose that represents a 21-joint configuration of a grasp type during a grasping task. (a) is the grasp pose for a prismatic-2-finger grasp (pinch grasp), and (b) represents a spherical grasp.

In applications to prosthetic grasping, a deep-learning model that first estimates grasps, as grasp types, is a simpler and less error-prone method than estimating the individual joint angles and positions of a hand. A default grasp pose with the 21-joint configuration can be assigned to each unique grasp type (Fig. 2.4), where each grasp pose is modifiable if a tighter fit on the selected object’s shape is necessary. In addition, the grasp-pose estimation model needs to be modified and re-trained each time a new hand with varying DOFs from the original hand is tested.

## 2.4 Affordance Detection

Every object in a scene can be interacted with in multiple ways depending on the chosen task. As humans, we intuitively understand what actions each object can afford, their connections with other objects, and the reasoning behind a certain grasp pose for the intended task. For example, if a whole apple and a knife are placed next to each other on top of a cutting board, in this scene, the apple affords being grasped for a cut or transfer task; the knife affords being used for a cutting task by holding its handle; and the cutting board affords to support both the apple and knife on its surface. The scene is in a kitchen, where the apple is deformable, and the knife is used to cut the apple; the action of cutting an apple can only be performed on the cutting board, which provides a flat surface to work on without damaging the countertop under the cutting board. In this situation, one of the user's hands will grasp the apple and expose the desired cuttable surface; the other hand will pick up the knife by its handle and perform the cutting task on the apple. Affordance detection therefore assists the selection of task-oriented grasps based on the predictable actions on the objects. Affordance detection also allows each object to be further segmented based on their object parts, and independently classified for their possible actions. However, an issue arises when the same object part or region contains multiple affordances. This problem is generally avoided by eliminating redundant affordance classes that are similar to existing classes. In the knife and apple example, the *cut*, *peel*, *chop* affordance classes all occur on the handle of a knife, which can be simplified to a single *use* affordance class. As described in Chapter 1, affordances are either represented as object affordances [8, 9, 10] or grasp affordances [11, 6, 5, 7, 12] in robotic applications. Affordance detection methods are only found in robotic grasping applications and have yet to be implemented for prosthetic grasping applications.

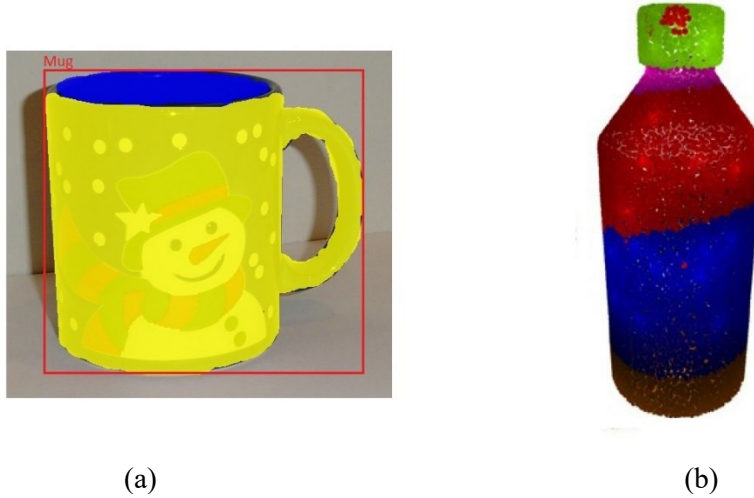
In current prosthetic grasping methods, a single grasp type is predicted for each object [25, 16, 17, 18, 26, 29, 19]. Myoelectric prostheses allows the user to select a fixed set of pre-programmed grasp types [16, 17, 18, 19], while vision-based prostheses (including vision-based myoelectric prostheses) use deep CNNs to predict a suitable grasp type that closely matches the shape of the selected object [25, 26, 29]. By adapting affordance detection in vision-based prostheses, the limitation in current prostheses of having no alternative and task-suitable grasp types can be solved. Alternative grasps are useful since the predicted grasp type can fail to pick up the object or is unsuited for a specific task (task-oriented grasps). For prosthetic grasping applications, the affordances can be detected as the grasp affordances: to identify all object parts in an RGB image and assign each object part with a grasp type. As seen in Sec. 1.3, most vision-based prostheses use a hand-mounted camera to capture RGB images of single objects. When the

camera is head-mounted instead, the field of view of the environment is expanded, and the affordance detection of multiple object scenes is possible. The head-mounted camera setup for prosthetic grasping applications can allow the detection of affordances on other objects, which can influence the resulting grasp-type predictions to grasp an object.

#### **2.4.1 Object Affordance Detection**

Object affordance detectors are all modelled as a multi-class classification problem, a regression problem, or both. When affordance detection is applied to 3D point clouds of objects, each data point is independently classified with an object-affordance label [49, 9]. When object-affordance detection is applied to 2D images and videos, models like AffordanceNet [8] apply image segmentation techniques on object parts with a per-pixel classifier. Fig. 2.5 provides an illustration of expected affordance detection results from ground truth annotations found in the IIT-AFF [8] and 3D Affordance Net [9] datasets. In both Figs. 2.5a and 2.5b, each object affordance region is represented with a different colour in the image. Fig. 2.5a illustrates a single-object RGB image of a mug with a bounding box and the 2D object affordance regions as image masks. Fig. 2.5b illustrates a 3D point cloud of a bottle, where each 3D point is assigned a different affordance label. In application to prosthetic grasping, only 2D RGB image affordance detection methods are considered in this thesis since it is assumed that a prosthesis user would only have access to a single FPV camera to capture scenes. As with AffordanceNet [8], the affordance detection model is applied to multiple-object RGB image scenes, where both the object classes and their respective object affordances are simultaneously estimated.

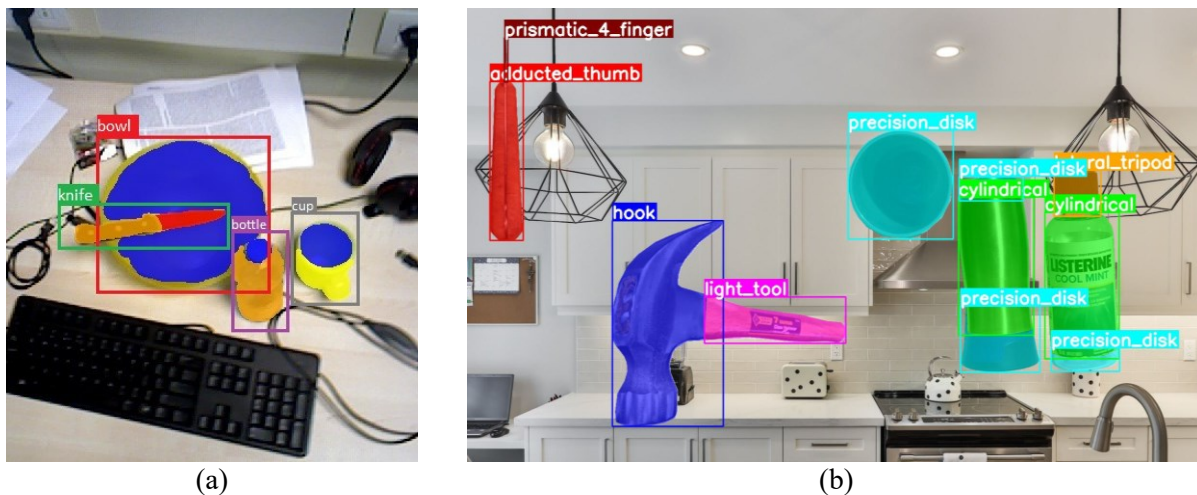




**Figure 2.5:** Ground truth affordances from (a) IIT-AFF and (b) 3D AffordanceNet datasets. (a), the wrap-grasp affordance (yellow) and the contain affordance (blue) are bounded by a bounding box of the detected mug. In (b), each 3D point in the point cloud is assigned a single affordance label: grasp (red), wrap-grasp (blue), contain (orange), open (green), pour (magenta).

AffordanceNet is a variant of the Mask R-CNN instance segmentation model [33] that requires two stages to make object affordance predictions from input RGB images. In the first stage, a variant of the Faster R-CNN [56] object detection model is used to segment and localize all image patches containing the Regions of Interest (ROI) of supposed object locations [8]. A VGG16 backbone extracts the object features and predicts the ROI regions as bounding box proposals through a Region Proposal Network (RPN) [8]. A ROI Align module then converts each ROI region into a  $7 \times 7$  feature map, where each ROI feature map is aligned with the full input image feature map [33, 8]. The resulting ROI feature maps are fed as inputs into the second-stage subnetworks of AffordanceNet. In the second stage, the affordance and object detection networks run in parallel with individual loss calculations, but the loss optimizations are jointly evaluated. The object classification network is comprised of a dual layered Fully Convolutional Network (FCN) [63] to further learn the features of the ROI feature maps. The extracted features are then converted to a  $N \times C$  vector of object class probabilities with a linear softmax layer [8], where  $N$  represents the number of instances in an image and  $C$  represents the probability score of an object class. In addition, the extracted features are regressed to produce  $N \times 4$  bounding box locations: the box’s  $x$ - $y$  centre coordinates, width, and height relative to the dimensions of the input image in pixels. This method allows all objects in a scene to be simultaneously detected and localized in a single pass. The affordance

subnetwork also feeds the same ROI feature maps, and the affordance subnetwork has a multi encoder-decoder structure that gradually upscales the feature maps to the same dimensions as the original RGB image [8]. The final output consists of all the detected bounding boxes of objects with the corresponding object classes, the classification scores, and the segmented affordance image masks. Unlike the traditional Mask R-CNN [33] instance segmentation model, the affordance subnetwork is a semantic segmentation model. The affordance image mask produced by AffordanceNet are 2D per-pixel affordance label maps with matching dimensions of the input images. Each pixel in the affordance label map can only be classified to a single affordance label; thus, overlapping affordance image mask predictions are not possible with AffordanceNet. If an object has overlapping affordance masks, then each of those affordance masks can be treated as an independent instance. Each instance has a binary mask against the background label (true if it contains the affordance label, false otherwise) with their corresponding affordance label and bounding box. Fig. 2.6 illustrates the difference between an affordance-image mask predicted as a per-pixel affordance map (Fig. 2.6a) and as instances of binary affordance masks (Fig. 2.6b).



**Figure 2.6:** Comparison of affordance masks as: (a) a per-pixel affordance label, and (b) as independent instances. (a) are the object affordances from the IIT-AFF dataset [8], and (b) are the grasp affordances (as grasp types) from the MOMA synthetic dataset. Note that the objects in (b) appear to be floating since the image was created by placing random single-object images on a random background.

### 2.4.2 Grasp Affordance Detection

Affordances can be interpreted as affordable grasp poses, as in the GanHand [5] model, to show all the possibilities of task-oriented grasp poses on an object. Each grasp affordance region is evaluated to determine the task-suitability pertaining to a certain grasp pose [50, 6, 10, 5, 7, 12]. In conjunction with the object or hand-object pose estimation models, affordance detection models are deployed first for scene understanding before a grasp pose is chosen for a desired task [49, 10, 7, 12]. Alternatively, grasp affordances can be represented as task-suitability regions with either segmentation [6, 11, 12] or object detection [7] methods. Additional features, including the environmental context and object attributes can be implemented on an affordance detection model and visualized in a Knowledge Base (KB) graph [7]. An example of environmental context includes the location of the visualized scene; and an example of object attributes includes its texture, material type, and its purpose (such as for food storage, or as a utensil) [7]. Since the GanHand [5] model theoretically detects grasp types as grasp affordances, then each object part can represent a different grasp-type region. Unlike the GanHand [5] model, more than one grasp type should exist on an object if applicable; for example, a bottle has three unique grasp affordances, as illustrated in Fig. 2.6b; a lateral tripod grasp for the bottle lid, a precision disk grasp for the base of the bottle, and a cylindrical wrap for the body of the bottle.

### 2.5 Task-Oriented Grasping

Traditionally in robotic systems, prehensile grasp poses are selected based on the overall stability of the grasp configuration, by evaluating its grasp quality index [43, 34, 35]. As described in Sec. 2.1.1, stability in a grasp pose is achieved by balancing all wrenches at the contacts to achieve force closure [35, 34, 21]. However, there is an infinite number of solutions to balance all contact wrenches to equilibrium. This is because each friction cone contains varying force directions within its extrema, and varying magnitudes of forces can be applied at each contact [34]. Therefore, the grasp quality index is utilized to solve an optimization problem to identify the most efficient combination of wrenches within the grasp wrench space (GWS), while minimizing the forces required to resist external wrenches and ensure force closure [43, 34, 35]. The grasp wrench space is the convex hull of all wrenches produced at the contacts [43, 34, 35], and a higher grasp quality index means that a grasp is more stable.

Although the estimation of stable grasp poses allows successful manipulation of the object, there is a lack of detail pertaining to how these defined grasps are tuned to complete a specific task. For example, a bottle can be grasped in many ways depending on the task: a cylindrical wrap-grasp parallel to the length

of the bottle for a pick-and-place task; and a tripod grasp facing downwards towards the table to transfer the bottle along a surface or to twist the bottle cap open. As seen in the YCB-Affordance Dataset [5], the selected grasp poses may not necessarily be the most stable; however, they are the most task-suitable pose.

Task-oriented grasping has been widely applied to robotic applications [61, 45, 11, 49, 46, 6, 5, 10, 7, 12, 8], but not yet towards prosthetic grasping. Task-oriented grasp poses are predicted in conjunction with affordance detection models, since affordance detection describes the possibilities of actions on an object [11, 6, 5, 10, 7, 12, 8]. The process of task selection is generally defined by the user, and the following affordance detection model determines the task-suitability locations for a task. The grasp poses can then be estimated based on learned object shapes, or hand-object poses, as seen in Sec. 2.3. Therefore, a new multimodal model can improve the performance of a vision-based prosthesis by leveraging task-oriented grasp poses (or grasp types) with affordances.

## 2.6 Summary

Current methods in vision-based prosthetic grasping predominately use a fixed camera on the prosthetic hand. The grasp selection method can therefore simplify to the single object within the camera frame. However, the extreme motion of the camera can lead to the capture of poor-quality image frames, where the captured objects could be occluded by other objects or the prosthetic hand in a pre-grasp shape. The object could also simply not be captured during the duration of the grasping task if the camera is not oriented to face the intended object. An FPV camera can resolve most of the issues found in an on-hand placement of a camera. The FPV camera allows the capture of multi-object cluttered scenes in the user's perspective, as seen in the analysis of prosthesis usage in upper limb amputees in Sec. 2.2.1.

Vision-based prosthetic grasping methods are currently limited to the classification and detection of a single grasp type from a single-object RGB image [29, 25]. By correlating a single grasp type to a single object, the classification model is unable to predict and use alternative grasp types in the case the prosthesis fails to pick up the object. The prediction of alternative grasp types is necessary if the predicted grasp type is misclassified and unsuitable for the object. As seen in Sec. 2.3, multiple attempts to predict a suitable grasp type on a single object are made in a single trial when a grasping task is executed. Each prediction attempt leads to a significant increase in total runtime, leading to the infeasibility of the grasp selection method [25]. As described in Sec. 2.4, alternative grasp types are also necessary to allow a prosthetic user to choose different grasp types on the same object for different tasks. For example, a

cylindrical grasp type can be used to hold the mug's body for the most stable pose, while a hook grasp on the mug's handle would permit drinking or pouring fluids. In current prosthetic grasping methods, multiple grasp types have been assigned to each unique object [27]. However, the prosthesis user still needs to proactively reject all unsuitable grasp types for the current grasping task [27], which also increases the runtime of the grasping task also increases [27]. As mentioned in Sec. 2.3.1, the main issue with current vision-based methods [27] is the assignment of grasp types to the whole object, instead of the individual object parts.

The current vision system in prosthetic hands can be improved to solve the aforementioned issues (i.e. limited task-oriented grasp selection and the one-to-one correspondence of an object with a grasp type), by adapting the robotic grasping methods discussed in Secs. 2.3 to 2.5. Recent robotic grasping applications introduce the estimation of stable prehensile grasps as a 6D grasp pose [49], a 21-jointed configuration grasp pose [49, 41], or the combination of both [41]. The estimation of grasp poses is directly correlated to an object's shape. Grasp poses are the hand poses that are specific to grasping, which can ultimately define a grasp type. By detecting the grasp types first, a unique robotic gripper or terminal device with varying DOFs could all be trained and tested on the same grasp-type estimation model. As seen in Sec. 2.3.2, grasp types are used in robotic grasping applications to first define the general shape of a grasp pose before the final grasp pose is adjusted to better fit the object's shape [5]. Since current vision-based prosthetic grasping methods already estimate grasps as grasp types [25, 27], the prosthetic grasping vision system developed in this thesis will also predict grasps as grasp types to simplify the grasp estimation model.

The robotics definition of a grasp pose is primarily based on a stability criterion that maximizes the overall grasp quality on an object [26, 34, 35]. However, the most stable grasp pose is not necessarily the most suitable grasp pose for a given task [11, 6, 5, 7, 12]. Currently, prosthetic grasping methods also prioritize stable grasp poses over task-suitable grasp poses [25, 26]. Therefore, in robotic applications, the affordances of an object (either as object affordances or grasp affordances) are widely used for choosing task-oriented grasps through scene understanding, by correlating the visual features of an object with semantic reasoning [6, 10, 7, 12]. The visual features of an object include the object's parts and the potential actions the object provides to the user. As seen in Sec. 2.4.2 and 2.5, the affordances of an object provide vital contextual information about where an object can be grasped for a specific task based on the predicted grasp affordance regions. Similar to robotic grasping applications, the detection of affordances

in prosthetic grasping can introduce a new method of assigning a task-oriented grasp type for each object part detected in an RGB image.

In current prosthetic grasping applications, affordance detection and task-oriented grasping are not applied in any vision-based prosthetic grasping models. Therefore, in this thesis, affordance detection models from robotic applications will be analyzed and adapted for prosthetic grasping applications, where the goal is to find the distribution of grasp types and the task-suitable regions in a multi-object, cluttered RGB image. Although the use of depth images with RGB images (i.e. multi-modal data) has improved the overall accuracy of classifying grasp types [29, 26], only RGB images will be used. All models analyzed in this thesis research exclusively accept RGB images as the inputs. Theoretically, task-oriented grasp types and potential tasks on an object can be estimated as grasp affordances and be incorporated into an affordance detection model. Each grasp affordance region on an object part will contain a suitable grasp type to grasp the object. When grasp affordances are detected as tasks, the grasp types detected on an object can be filtered to match the potential tasks on an object. By combining the detection of grasp affordances as grasp-types and task-regions, the prosthetic hand user will be able to choose a selection of related grasp types that are suitable for the task. Therefore, in this thesis research, a multi-grasp affordance model can be proposed to improve the vision system in prosthetic hands.

## Chapter 3

### Methods for Simultaneously Detecting Objects and Their Grasp Affordances with A Deep CNN Segmentation Model

In this research, an ablation study (discussed in Sec. 3.1) was initially conducted to determine the best method for detecting objects (i.e. cups, bottles, knives, and hammers) and their grasp affordances in RGB images. In the context of machine learning, an ablation study is an experimental method, which involves the removal of a module in a machine learning model to understand the significance of the module [64]. A module is essential to a machine learning model if the machine learning model performs worse (e.g. lower classification accuracy) without the module. Since most object affordance detection methods use segmentation models (as described in Sec. 2.4.1), the ablation study in Sec. 3.1 will investigate the significance of the object detection module in instance segmentation. Traditionally, an instance segmentation model detects objects (cups, bottles, etc.) as instances of an image mask and a localized bounding box for a given RGB image. Each of the detected instances corresponds to a single object class. A semantic segmentation model is similar to an instance segmentation model, except that the semantic segmentation model lacks an object detection module. For a given RGB image, a semantic segmentation model predicts an object class for each pixel in the image. Therefore, the image mask for each object class is present; however, the location of each object in an RGB image is not explicitly determined. For the ablation study, the object affordances will be detected instead of the objects.

The ablation study in Sec. 3.1 found that an instance segmentation model was significantly more accurate in detecting object affordance masks than a semantic segmentation model. In AffordanceNet [8], an instance segmentation model, the resulting object affordances were precisely localized in the bounding boxes of the detected object in an image. Unfortunately, most existing instance segmentation models are only trained to detect the instances of objects (cups, bottles, etc.) from the COCO dataset [65]. Therefore, a new Multi-Object Multi-Grasp-Affordance (MOMA) synthetic dataset was created in this research to allow the simultaneous detections of objects, grasp types, and task-region as instances. The details of the MOMA dataset creation are described in Secs. 3.2 and 3.3. Secs. 3.4 and 3.5 discuss how the instance segmentation models were trained and evaluated in Chapter 4. In addition, Secs. 3.4 and 3.5 introduce the newly developed Multi-Affordance Detection Network (MAD-Net) for the joint detection of objects and their grasp affordances as instances.

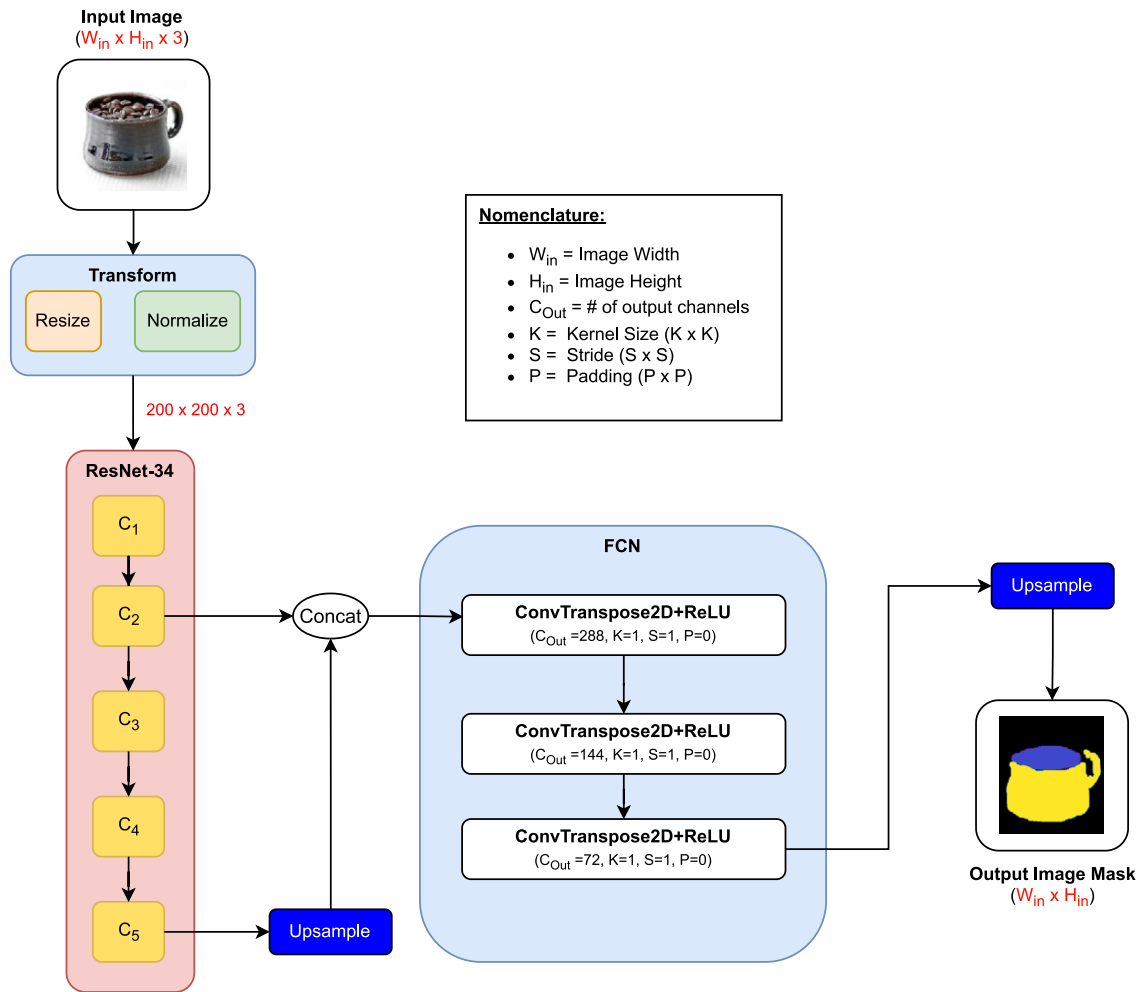
### **3.1 Ablation Study to Determine the Significance of Object Localization in Instance Segmentation Models**

The purpose of the following ablation study was to determine the significance of object localization in instance segmentation models, by removing the object detection module from the studied model. An object detection module provides the location of a labelled bounding box in an RGB image to localize and classify an object. The instance segmentation model investigated in this ablation study is the AffordanceNet [8] model, which has a backbone network for feature extraction (typically a variant of the ResNet model [66]), an object detection module, and an image segmentation module. The ResNet-34 FCN model was developed to compare the object affordance detection results with the AffordanceNet model [8]. The dataset used to train the ResNet-34 FCN and AffordanceNet [8] models for object affordance detection is the IIT-AFF [8] dataset, which provides the annotations for 10 object-affordance classes on real RGB images. Currently, there is no dataset that provides the annotations to simultaneously detect objects, grasp types, and task regions as bounding box and image mask segmentations. The outcome of the ablation study would dictate the type of dataset that needs to be created to train the object and grasp affordance detection model.

#### **3.1.1 ResNet-34 FCN Semantic Segmentation Model Architecture**

In the ablation study, a supervised semantic segmentation CNN model, ResNet-34 FCN, was trained and tested on the IIT-AFF RGB [8] image dataset. The CNN model followed the encoder-decoder architecture, since it adapts the Residual Network (ResNet) [66] architecture as the backbone for feature extraction, where a proceeding FCN-like subnetwork upsamples the features back to the original input image dimensions with the object affordance scores. Fig. 3.1 shows the architecture of the ResNet-34 FCN network. The ResNet model [66] used is a deep CNN classification model with 34 convolutional layers (ResNet-34) pre-trained on the ImageNet-1K dataset [67]. ImageNet-1K contains 1000 different object classes with over one million images in the training dataset [67].





**Figure 3.1:** Network Architecture of the ResNet-34 FCN model for object affordance detection. The *Concat* blocks concatenate the feature maps along the channel dimension.

A deeper CNN model with more convolutional layers usually results in lower training and validation accuracy; however, the performance of the Res-Net model improves as more convolutional layers are added to the model. Beginning at the outputs of the first convolutional block to the outputs of the last convolutional block, a skip connection is successively added to every pair of convolutional layers. The skip connection takes each pair of convolutional layers and concatenates them with the previous pair of convolutional layers, through element-wise summation. As a result, a CNN model with skip connections can recall and learn the features from the earlier layers of the CNN, where vanishing gradients are mitigated. Vanishing gradients occur when a model's weights near the input layers of a neural network converge to zero during backpropagation (the backward pass during weight optimization), where the

neurons at the earlier layers of the model are unable to update their values and the model thus cannot learn the features from the earlier layers. The ResNet model also includes a batch normalization layer in between each convolutional layer and Rectified Linear Unit (ReLU) activation function, which also solves the vanishing gradient problem [63].

Following the ResNet-34 backbone for feature extraction, a decoder with an FCN-like CNN subnetwork gradually upsamples the resolution of the feature maps to a close approximation of the original input image dimensions. The decoder consists of three deconvolutional layers, where the output channels of each succeeding deconvolutional layers are reduced by half, from 576 channels to 72 channels; the deconvolutional layers are each activated with a ReLU function. Each deconvolutional layer contains trainable weights that use  $1 \times 1$  kernels to simulate the one-to-one connections of each neuron in the model network as seen in the FCN [63] model. When  $1 \times 1$  kernels are used to convolve the oncoming inputs, the resulting output feature map has the same dimensions as the input feature map. In the final layer of the ResNet-34 FCN model, the outputs of the decoder are upsampled once more, to ensure the outputs match the input image dimensions. The outputs of the decoder are upsampled with a bilinear interpolation layer that does not contain trainable weights. The ResNet-34 FCN model also has a skip connection to connect the outputs of the second convolutional block in the ResNet-34 backbone to the first deconvolutional layer. The skip connection concatenates the two layers by stacking the feature maps along the channel dimension, instead of element-wise summation.

The outputs to the ResNet-34 FCN model contains pixel-wise object affordance class scores of size  $N \times C \times H_{in} \times W_{in}$ , where  $N$  is the batch index,  $C$  is the total number of object affordance classes in the IIT-AFF dataset (which is 10),  $H_{in}$  is the height of the input image, and  $W_{in}$  is the width of the input image. Each pixel has 10 object affordance class scores of raw probabilities ranging from 0-1. However, for each pixel, the total sum of raw probabilities across all object affordance classes may not necessarily sum to one. The raw probabilities in this case are known as the logits.

### 3.1.2 Training and Evaluation of the ResNet-34 FCN Semantic Segmentation Model

The ResNet-34 FCN model was trained and evaluated in PyTorch [68] on the IIT-AFF [8] dataset, which contains 8835 real RGB images of single-object and multi-object scenes, with some image sequences in an egocentric view. Each image contains an annotation for a per-pixel object affordance label map, a per-pixel depth label map, and object classes with bounding boxes in  $(c_{obj}, x_{min}, y_{min}, x_{max}, y_{max})$  format.  $c_{obj}$  is the object class label,  $(x_{min}, y_{min})$  are the top-left box coordinates in pixels, and  $(x_{max}, y_{max})$  are

the bottom-right box coordinates in pixels. In this ablation study, only the object affordance map annotations were used since object detection was not required. There were ten object classes and ten object affordance classes in the IIT-AFF dataset. The object classes consisted of the following labelled from 0-9 in the following order: bowl, TV monitor, pan, hammer, knife, cup, drill, racket, spatula, and bottle. The object affordance classes consisted of the following labelled from 0-9 in the following order: background (no object affordance), contain, cut, display, engine (the authors of the IIT-AFF dataset [8] have only used this object affordance to indicate the motor on a drill), grasp, hit, pound, support, and wrap-grasp (form closure grasps). The IIT-AFF dataset was partitioned, where 70% of the images were for training (6184 images) and 30% of the images for testing (2651 images) the ResNet-34 FCN model. A validation set was not included in this ablation study since hyperparameter tuning was not necessary. The evaluation results (Sec. 3.1.3) of the model were sufficient to determine the feasibility of using semantic segmentation for affordance detection without object localization (as in affordance segmentation).

During training, the input images were all equally resized to  $200 \times 200$  images, and each pixel was normalized to values between 0-1, with the RGB mean  $[0.485, 0.456, 0.406]$ , and standard deviation  $[0.229, 0.224, 0.225]$ , values from the ImageNet dataset [67]. Each of the normalized, resized input images were converted to tensors (n-dimensional arrays) for computations in the GPU. The target object affordance map (ground truth) was also converted to a  $H_{in} \times W_{in}$  tensor with the same height and width dimensions as the input image. The ResNet-34 FCN model was trained with a batch size of 64 images and optimized with the Adaptive Moment Estimation (ADAM) optimizer [69], at a learning rate of  $1 \times 10^{-3}$ . The model needed 6.8 GB of GPU memory, and 12 hours and 46 minutes to train for 100 epochs on a Nvidia GeForce RTX 3080 card. An epoch represents the evolutionary state of the model parameters (such as the model weights), where each training image has, at least once, been processed by the model. In this ablation study, the ResNet-34 FCN model was trained beyond 10 epochs to examine the trends in the training loss curve beyond the initial point of convergence with the minimized loss value. Throughout training, the model was optimized with the cross-entropy loss function, which is the negative log-likelihood loss as a function of the output logits as log-softmax probabilities, as represented in the following equations [68].

$$S(\mathbf{X}_n, \mathbf{Y}_n) = \frac{\exp(\mathbf{X}_n, \mathbf{Y}_n)}{\sum_{c=c_0}^C \exp(\mathbf{X}_n, c)} \quad (3.1)$$

$$\mathbf{L}_{CE_n} = -\ln S(\mathbf{X}_{n,\mathbf{Y}_n}) \quad (3.2)$$

$$L_{CE} = \sum_{n=1}^N \frac{L_{CE_n}}{\text{count}(\mathbf{Y}_n)} = \sum_{n=1}^N \frac{\sum_{i,j} (\mathbf{L}_{CE_n})_{ij}}{\text{count}(\mathbf{Y}_n)} \quad (3.3)$$

$\mathbf{X}_{n,\mathbf{Y}_n}$  is a subset of the predicted output logits from the ResNet-34 FCN model,  $\mathbf{X}_n$ , for the positive target object affordance class in the target object affordance tensor,  $\mathbf{Y}_n$ .  $\mathbf{X}_{n,\mathbf{Y}_n}$  is a tensor with the input image dimensions,  $H_n \times W_n$ , of image  $n$  in a batch of  $N$  images. Each pixel value in  $\mathbf{X}_{n,\mathbf{Y}_n}$  is assigned to one of the 10 output logits in  $\mathbf{X}_n$  (each pixel in  $\mathbf{X}_n$  is a  $10 \times 1$ , 1D vector) based on the matching pixel location and value of the target object affordance class. As an example, a pixel located at  $(1,0)$  has a target object affordance class of three. Since the pixel has 10 output logits, only the fourth output logit is assigned to the pixel (since the object affordance classes are indexed from zero). Eq. 3.1 is the softmax function with the same dimensions as  $\mathbf{X}_{n,\mathbf{Y}_n}$  that bounds the output logits to probability values that sum to 1 across all object affordance classes. The softmax function is the ratio between  $\mathbf{X}_{n,\mathbf{Y}_n}$  and the natural exponent of  $\mathbf{X}_n$  that is summed across each object affordance classes,  $c$ .  $c$  is an integer belonging to the 10 possible object affordance classes, where  $[0 \leq c < 10]$ . Eq. 3.3 is the total cross-entropy loss for all predictions in an image batch  $N$ , where  $\mathbf{L}_{CE_n}$  (Eq. 3.2) is a tensor of pixel-wise cross-entropy losses for a single image with matching dimensions as  $\mathbf{X}_{n,\mathbf{Y}_n}$ . The negative operator is added to Eq. 3.2 since the natural logarithm of a probability value is negative, and the loss value is generally set to positive. The total cross-entropy loss for a batch of  $N$  images is reduced to a single loss value with Eq. 3.3: the cross-entropy loss tensor for each image  $n$  is element-wisely summed and averaged with the total number of elements in the target object affordance tensor (average loss per pixel). The cross-entropy loss value per image is further summed to obtain the total cross-entropy loss for the batch of images.

Before evaluation, the predicted output logits of ResNet-34 FCN, for each image,  $\mathbf{X}_n$ , were first converted to an object affordance map as follows:

$$f_{aff}(\mathbf{X}_n) = \arg \max_c \mathbf{X}_n \quad (3.4)$$

where  $\mathbf{X}_n$ , with the dimensions of  $C \times H_{in} \times W_{in}$ , is converted to a  $H_{in} \times W_{in}$  matrix, where each pixel is assigned to an index value of  $c$  (the object affordance label) that contains the highest object affordance class scores along dimension  $C$ . For example, a pixel in the output object affordance map has a  $10 \times 1$ , 1D vector of object affordance class scores, where the highest score is located at index two. Therefore, the

object affordance class at that pixel is also two, which is the third object affordance class (since  $c$  and the index position starts at zero). Each pixel in the predicted and target object affordance map can only contain one value belonging to  $c$ .

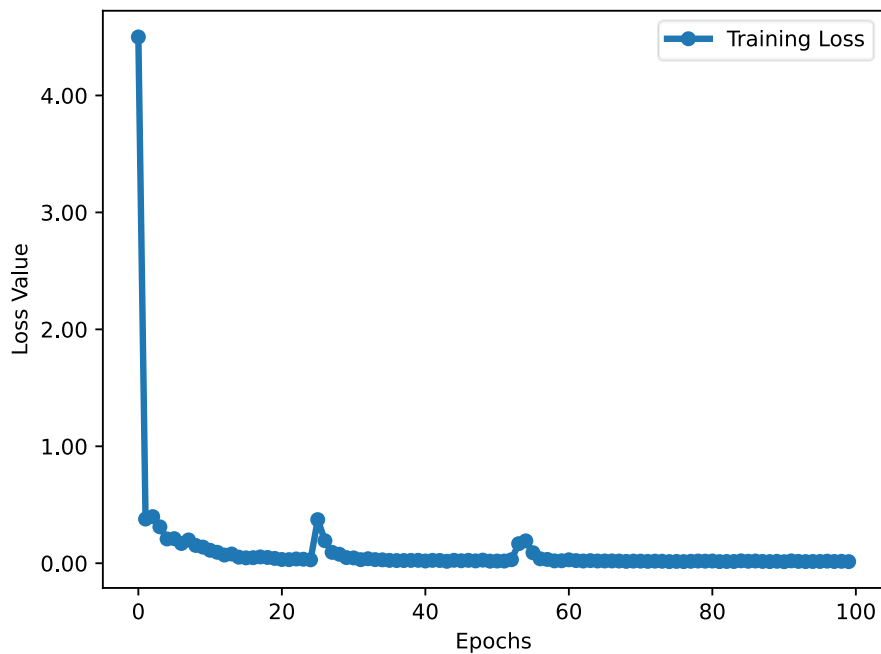
The mean intersection over union (mIOU) metric was used to evaluate the performance of the ResNet-34 FCN model on the training and test sets of the IIT-AFF dataset. A mIOU value was calculated for each test image in the test set and averaged across all the images in the test set. The mIOU metric indicates how accurate the predictions (the segmented object affordance masks) match with the ground truth (the target object affordance masks) by evaluating the total class-based ratios of overlapping area between the predictions and the ground truths. A model that is evaluated on a dataset performs well if it can achieve an mIOU score of more than 0.5, where 1.0 is a perfect score. As seen in the equation below [70], the overall mIOU is the average of all IOUs (intersection over union) from each class  $c$ , where mIOU can have a score from 0-1.

$$\text{mIOU}_n = \frac{1}{C} \sum_{c=c_0}^C \frac{\text{area}(f_{aff}(\mathbf{X}_{n,c}) \cap \mathbf{Y}_{n,c})}{\text{area}(f_{aff}(\mathbf{X}_{n,c}) \cup \mathbf{Y}_{n,c})} \quad (3.5)$$

$\mathbf{X}_{n,c}$  and  $\mathbf{Y}_{n,c}$  represent the segmented image masks of all pixels that have an object affordance class  $c$ .  $\mathbf{X}_n$  is the predicted object affordance map calculated from Eq. 3.4 and  $\mathbf{Y}_n$  is the target object affordance map.  $\mathbf{X}_n$  and  $\mathbf{Y}_n$  both have the input image dimensions of image  $n$ ,  $H_{in} \times W_{in}$ , containing all per-pixel labels for all possible values of  $c$  (integer values between 0-9, with 10 possible values). For each object affordance class, the local mIOU value calculated in Eq. 3.5, is the ratio of the intersecting area over the unified area between the predicted and target object affordance maps.

### 3.1.3 Results and Discussion of the ResNet-34 FCN Semantic Segmentation Model

The ResNet-34 FCN model achieved an mIOU of 0.884 on the entire training dataset, while the mIOU score was 0.544 on the entire test dataset. Therefore, ResNet-34 was evidently overfitting with the training dataset and cannot generalize to new unseen RGB scenes. The overfitting problem in the ResNet-34 FCN model can be attributed to the extended training of 100 epochs, when 10 epochs is sufficient to minimize the training losses. In Fig. 3.2, the training losses begin to converge to a minimum, as early as in the 10th epoch, and settle to a loss value of approximately 0.03 by the 20th epoch. There are also anomalies at the 25th and 53rd epochs as the loss values suddenly increase; however, the losses still eventually converge to a value of 0.01 by the end of the 100th epoch. The anomalies in the training loss were most likely due to the errors in weight optimization during backpropagation, which caused the training losses to increase before gradually converging to zero in the later epochs. Since the training losses converge, the model can make accurate predictions of object affordance maps that are similar to the ground truths, as reflected in the overall training mIOU score.



**Figure 3.2:** Training losses from the ResNet-34 FCN model for 100 epochs.

In the prediction images (Fig. 3.3), most of the pixel accuracy is attributed to the accurate classification of the background object affordance class 0. Specifically, the ResNet-34 model generalizes to the background where object affordances are non-existent. Since most of the RGB images in the IIT-AFF dataset predominately contain the background and not the objects, the mIOU metric is a poor indicator of performance for affordance detection. Affordance detection requires a model that emphasizes the accurate detection of objects, since an object's shape is the determining factor for predicting the varying affordances on an object. As seen in Fig. 3.4, the sample predictions of AffordanceNet [8] on the IIT-AFF dataset show that the added object localization of an object for affordance detection essentially removes most of the unwanted background noise in an image. Only the relevant pixels on the objects of interest, restricted by the detected object's bounding box, should be labelled with a non-background object affordance class. Therefore, instances of objects should be detected (predicted and localized object in an image) first before affordance segmentation should take place; an instance segmentation model like Mask R-CNN [33] can accomplish such task. In addition, an evaluation metric like the mean average precision (mAP) score rewards detection and segmentation models with better object localization accuracy; the evaluation method using the mAP metric will be discussed later in Sec. 3.4.2.

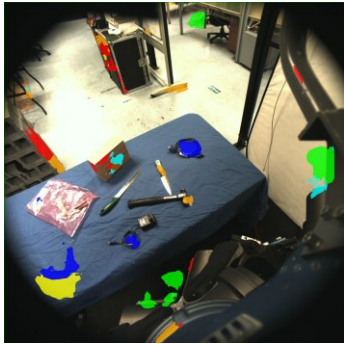
**Legend:** Contain ■ Cut ■ Display ■ Engine ■ Grasp ■ Hit ■ Pound ■ Support ■ Wrap-Grasp ■



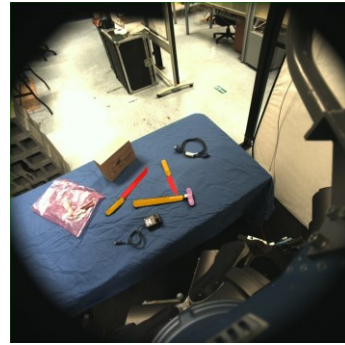
(a)



(b)



(c)



(d)



(e)



(f)



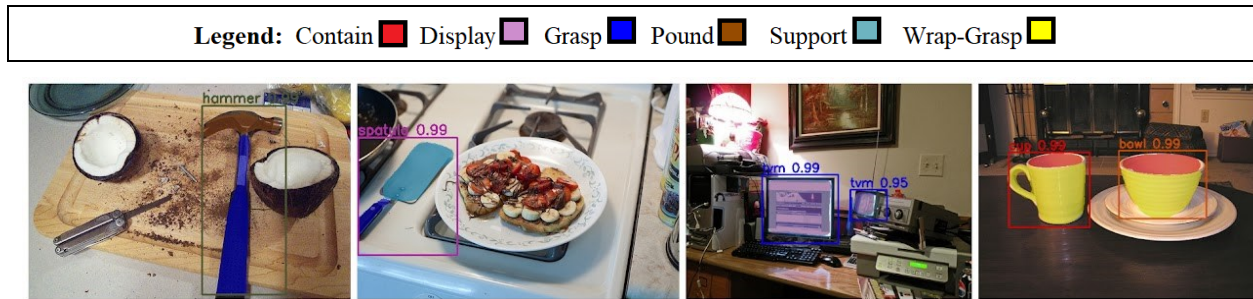
(g)



(h)

**Figure 3.3:** Inferencing results on the IIT-AFF [8] dataset for the object affordances (see colour key at the top of the figure). The images on the left are the predictions, and the images on the right are the ground truth annotations.





**Figure 3.4:** Sample inferencing results in the IIT-AFF dataset by the original creators of AffordanceNet, adapted from [8].

In this thesis, the methods used in this chapter are similar to how affordances were detected in AffordanceNet [91]. However, instead of predicting a single affordance map for the entire input image, each predicted affordance on an object part will have its own affordance map as instances of a binary image mask and a corresponding bounding box. Each binary mask contains a single affordance class, where pixels predicted with a value of one contain the affordance class, and the background for the value of zero. Therefore, each affordance and object are independently detected. In addition, each detected object will also include a segmented object image mask as instances. Since the purpose of this thesis research is to improve how grasps are detected for a vision-based prosthesis, grasp affordances will be detected instead of the object affordances. The grasp affordances are now defined here as the instances of the task-oriented-grasp-type and the task regions for task-suitability on the detected objects, where each grasp affordance represents the location an object affords to grasp by a prosthetic hand.

Currently, there are no available datasets that detect objects and their grasp affordances as instances of grasp-types or task-suitable regions. Instead of collecting and manually annotating real RGB images to produce a similarly sized dataset like IIT-AFF [8], a new dataset was synthetically generated as part of this thesis research. Theoretically, an infinite number of unique multi-object images can be created from a small subset of manually annotated single-object images [71]. However, there is an apparent reality gap in a model that is strictly trained on synthetic images, where the model's performance significantly decreases when evaluated with real images. Fortunately, models that are trained with a combination of synthetic and real data can still outperform models purely trained on real images [71]. Since the primary focus is to test the feasibility of adapting instance segmentation models for affordance detection, the choice of using real or synthetic data is irrelevant. A model can always be further trained with additional

real data to improve performance. The details about how the MOMA dataset was created will be discussed in Sec. 3.2 and 3.3.

### **3.2 Preparation of Single-Object RGB Images for Synthetic Dataset Generation**

The MOMA dataset has four primary object classes: cups (including mugs), bottles, knives, hammers. As outlined below, there are five steps required to prepare the single-object scenes for the synthetic dataset generation of the MOMA dataset:

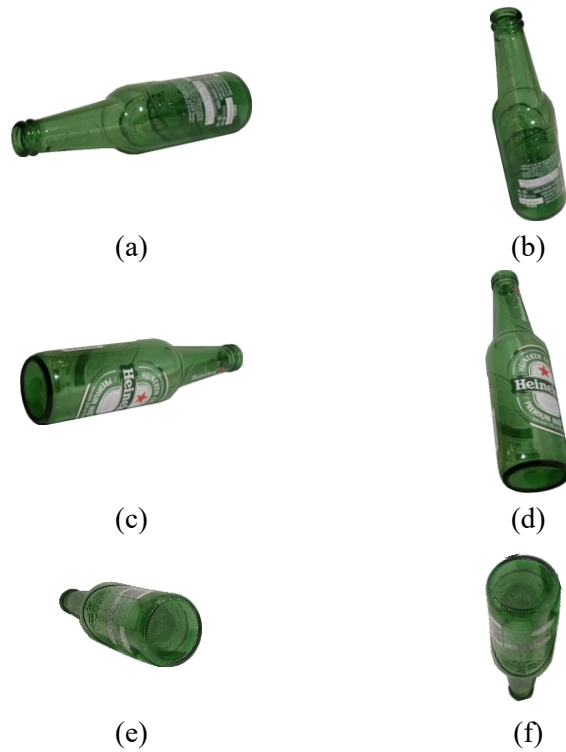
- a) RGB image collection of single-object scenes.
- b) Annotate each object in the RGB images as a binary image mask for background removal and as polygons with the x-y coordinates of the object's external edges.
- c) Annotate the grasp affordances as grasp-type regions by further partitioning the object's polygon annotation. Each grasp-type region is a polygon annotation on an object part.
- d) Additionally annotate the grasp affordances as task regions by combining the polygons of the object parts.
- e) Organize the images and annotations of each captured object into standard and irregular object sets. The standard object set contains objects with similar features (appearance, shape, functions) that are expected in one of the four primary object classes (cups, bottles, knives, hammers). Otherwise, the captured object is categorized into the irregular object set. The criteria for organizing the captured objects into these object sets are detailed in Sec. 3.2.5.

### 3.2.1 Image Collection of Real Objects in Single-Object RGB Scenes

Real RGB images of single-object scenes were captured using a Luxonis OAK-D Camera [72, 73] that was fixed on a tripod (Fig. 3.5). The OAK-D camera was set to UVC mode, to allow image capture through Windows 10's Camera application. Each image captured had a dimension of  $1920 \times 1080$  pixels, or a two-million-megapixel resolution. During image capture, the camera pose was adjusted to change the line of sight on an object, thus simulating the changing egocentric viewpoints of an observer. For all knives and hammers, three different camera poses were used since the knives and hammers could only be placed flat on a surface. Table 3.1 shows the different camera poses used to capture the objects. On the other hand, all bottles and cups were captured with the same camera pose since each image can be rotated  $90^\circ$  to produce an artificial egocentric viewpoint. The added rotation simulates the changing angle of elevation or depression when observing an object at different positions (Fig. 3.6).



**Figure 3.5:** Camera setup to capture real single-object RGB images with the OAK-D Camera.



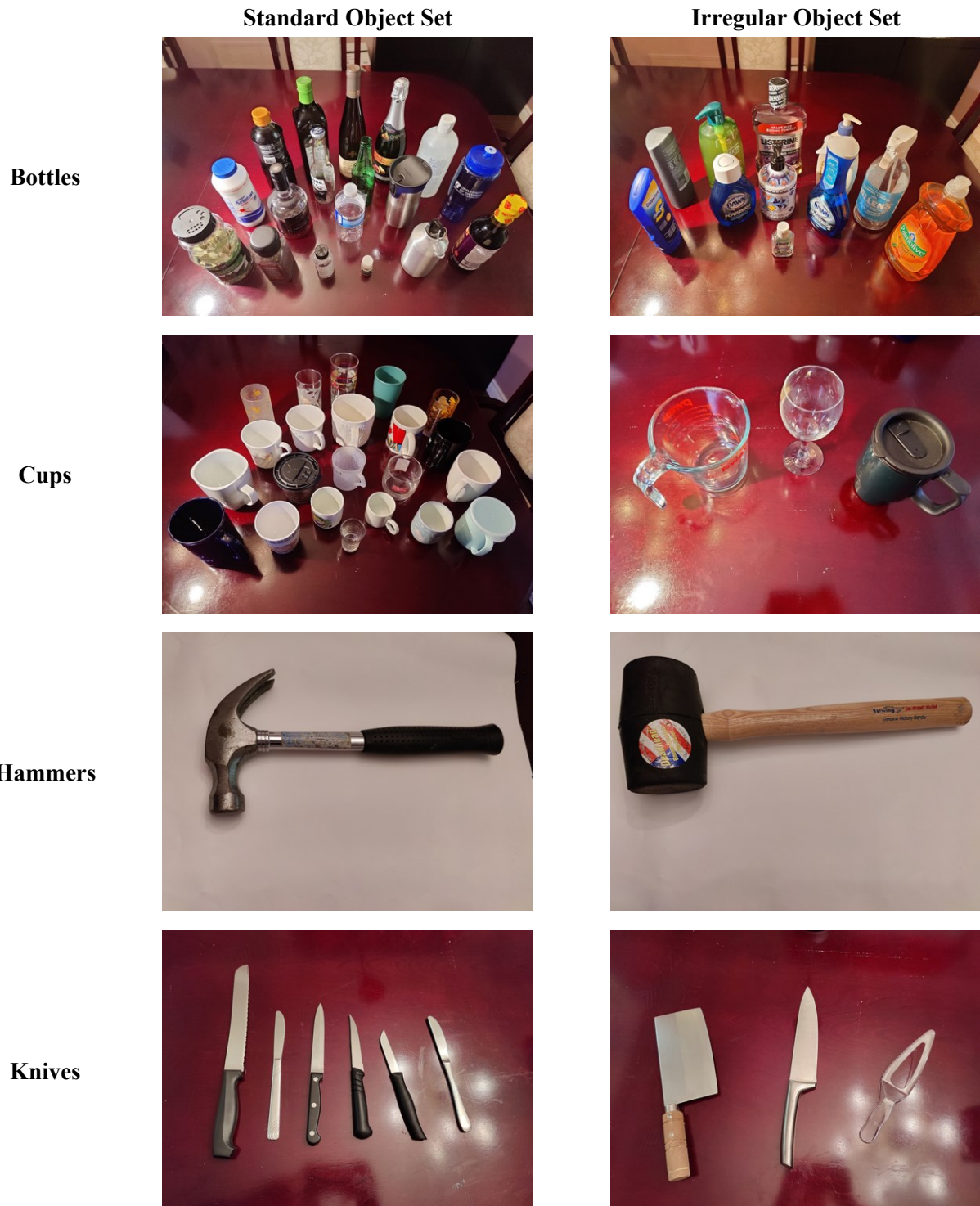
**Figure 3.6:** (a, c, e) original images of the captured bottle, and (b, d, f) the artificial change in the egocentric view of the object after a 90° rotation CCW or CW.

**Table 3.1:** Camera positioning during real image capture of objects.

	<b>Camera Pose</b>			
<b>Measurements</b>	1	2	3	4
<b>Objects Captured</b>	Bottles, Cups	Knives, Hammers	Knives, Hammers	Knives, Hammers
<b>Distance from camera to the object (along the diagonal)</b>	36 cm	30 cm	44 cm	75 cm
<b>Height of camera to table</b>	12 cm	16 cm	30 cm	70 cm
<b>Camera Tilt (w.r.t to the horizontal plane)</b>	0°	32.2°	43.0°	69.0°

Each object was placed on a white backdrop, and the objects were illuminated with a table lamp to ensure that the object was visibly clear without colour contamination from nearby surfaces and objects, especially if the object was transparent, or the object had a shiny surface. The distinctive colour contrast between the background and the objects assisted the background removal process during image annotation. For each object, an image was captured at every 45° increments as the object rotated about the axis orthogonal to the horizontal tabletop (azimuth). If the object could be stably placed in alternate poses by flipping the object on its side or bottom, then the same procedure of rotating an object per image capture was repeated. If the object was axially symmetric, then the similar or duplicate object poses would not be captured by the camera. On average, 12 images for each bottle, 50 images for each cup with handles (mugs), 12 images for each cup without handles, 24 images for each knife, and 24 images for each hammer were captured. The visible differences when changing the orientation of a mug's handle are considered, therefore each mug had four times more unique poses than the other object classes.

In total, 1561 images of 65 unique objects were captured, consisting of 29 bottles, 25 cups, 9 knives, and 2 hammers. Since there was an evident object-class imbalance between the four types of objects, additional knives and hammers were captured using 3D models, as described in Sec. 3.2.2. Fig. 3.7 shows the various objects used to collect the images for the real portion of the single-object dataset. As shown in Fig. 3.7, the objects captured for the MOMA dataset are organized into standard and irregular object sets. The exact categorization and reasoning for the partitioning of the objects are described in Sec. 3.2.5.



**Figure 3.7:** All real objects captured for the MOMA dataset, by object class and object set.

### 3.2.2 Image Collection of Synthetic 3D Objects in Single-Object RGB Scenes

The 3D object models of knives and hammers were obtained from Sketchfab, an online open-source repository of virtual 3D models. The collected 3D object models were uploaded to a WebGL GLTF viewer [74] and placed in a plain white background. A similar image collection method from Sec. 3.2.1 was used, where each 3D object was rotated in increments of  $45^\circ$  about the axis orthogonal to the horizontal plane. Each of the 3D objects were additionally adjusted to three different viewpoints by modifying the angle of elevation or depression, to simulate an observer looking up or down at an object that is placed on a surface, respectively. A total of 24 images were saved for each 3D object.

Occasionally, the 3D models were modified in Blender, a 3D modelling software, if the 3D object was not exactly aligned flat against the horizontal plane. Some of these unaligned models were left unmodified to capture the 3D objects in a unique pose that is unseen in the real single-object images.

Overall, 1216 images of 46 unique synthetic 3D objects were captured, consisting of 22 hammers and 24 knives. The images of the synthetic 3D objects were combined with the images of the real objects for a combined total of 2777 single-object RGB images. Fig. 3.8 shows the various 3D objects used to collect the images for the synthetic portion of the single-object dataset. Similar to the real images captured in Sec. 3.2.1, the objects were also partitioned into standard and irregular object sets, as described in Sec. 3.2.5.

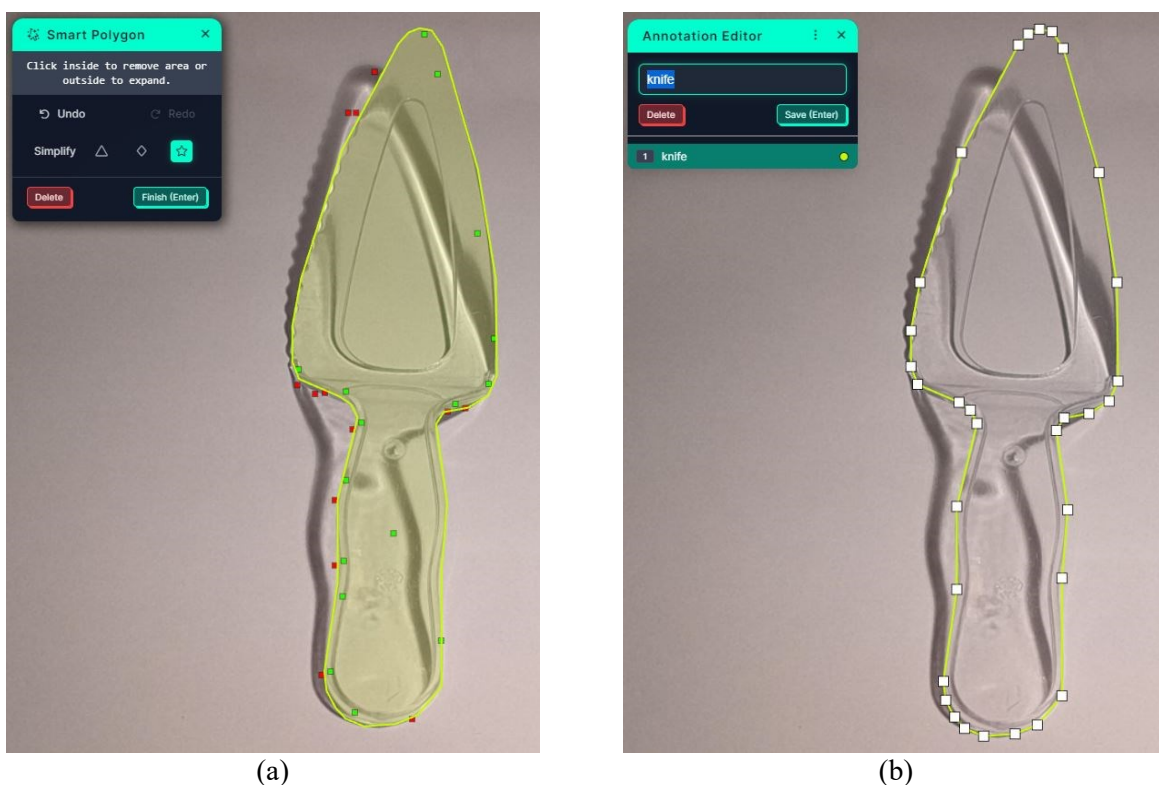




**Figure 3.8:** All synthetic objects captured for the MOMA dataset, by object class and object set.

### 3.2.3 Preparation of the Single-Object RGB Images for Object Mask Annotations

The single-object RGB images collected from Sec. 3.2.1 and Sec. 3.2.2 were first annotated for the object masks. Object-mask annotations are necessary for grasp affordance annotations, background removal in single-object images, and model evaluation. The real images were manually annotated with Roboflow [75], while the synthetic images were semi-automatically annotated by binary thresholding the image and manually adjusting the resulting object image masks with LabelMe [76]. Both Roboflow [75] and LabelMe [76] are annotation tools for drawing image masks and boxes as a series of x-y coordinates to form a polygon mask (Fig. 3.9).



**Figure 3.9:** Example polygon annotation process on a knife in Roboflow [75]. (a) creation of object image mask with the Smart Polygon tool, and (b) converted polygon annotation.

For the real images, binary thresholding cannot be applied since pixel noise is prevalent, despite having the objects placed on a white backdrop during image capturing. Unlike the synthetic images, the background of the real images is not a solid colour. However, the contrast between the foreground object



and background is significant enough to use the Smart Polygon tool in Roboflow, where the edge locations of the objects are automatically inferred. As seen in Fig. 3.9a, the green points represent the locations on an object, and the red points represent the background. The image mask then gets converted into x-y points (Fig. 3.9b). Once the polygon annotation is created for an object, the annotated objects were manually modified to ensure that the correct object edge was annotated, in the case of visible shadows, low contrasting edges of an object with the background (including transparent objects), or thin objects in the image. Each of the polygon annotations was labelled with the corresponding object class. Once all of the real images were annotated for objects as polygons, the dataset was converted to a single COCO JSON file. The COCO JSON file contains all of the images' metadata (such as the file location and image dimensions), and the corresponding polygon annotations. Since the generation of the synthetic dataset requires an individual annotation file for each image, the COCO JSON file was converted to a LabelMe JSON file for each corresponding image. The initial annotation was in COCO JSON format because Roboflow does not support the exportation of individual annotation files in LabelMe JSON format.

Using the OpenCV Python library [77], the synthetic images were first converted into binary image masks from an RGB image (Fig. 3.10b) by using Algorithm 3.1. Before binary thresholding was applied, the synthetic images were converted to a grayscale image. The three colour channels of an RGB image were compressed into a single colour channel of pixel intensities, where each pixel was represented as a single integer value between 0 and 255. The value of a grayscale pixel was obtained by calculating the weighted sum across the RGB channels for that pixel. To convert a grayscale image into a binary image mask, all pixel values above a certain threshold are set to a value of 0 (black), and 255 (white) otherwise. In this case, all pixels containing the object have a value of 255, whereas all background pixels have a value of 0. Occasionally, the segmented object may contain holes in the binary image mask, as seen in Fig. 3.10b. Therefore, the morphological closing transformation, which is a morphological erosion operation followed by a morphological dilation operation, is applied on the binary image mask to ensure that the external edge of the segmented object is more visually distinctive. The morphological erosion first removes the pixelated noise within the object, while the morphological dilation thickens the edges of the segmented object. Morphological dilation is necessary since morphological erosion shrinks the area of the segmented object mask. The morphological dilation and erosion operations are the respective Minkowski sums (in geometry, it is the sum of the points between two shapes [78]) and differences that are iteratively calculated by sliding a smoothing kernel across the entire binary image mask.

---

**Algorithm 3.1:** Conversion of a single-object RGB image to a binary mask.

---

CONVERT-TO-GRAYSCALE( $X$ )

**in:** an RGB image of 8-bit integer pixel intensities  $X \in \mathbb{N}^{H \times W \times 3}$  with height  $H$ , width  $W$ , and 3 colour channels, where  $\{X_{i,j,k} \in \mathbb{N} | X_{i,j,k} \leq 255\}$ .

**out:** a grayscale image of 8-bit integer pixel intensities  $Y \in \mathbb{N}^{H \times W}$  with height  $H$ , width  $W$ , where  $Y_{i,j} \leq 255$ .

```
1:   $Y \in \mathbb{N}^{H \times W} \leftarrow 0$                                 ▷ initialize a  $H \times W$  matrix of zeros
2:  for all  $\text{domain}(X, 3) \in X$  do                            ▷ iterate through each RGB pixel
3:     $Y_{i,j} \leftarrow 0.299c_0 + 0.587c_1 + 0.114c_2$         ▷ linear combination of RGB channels
4:  return  $Y$ 
```

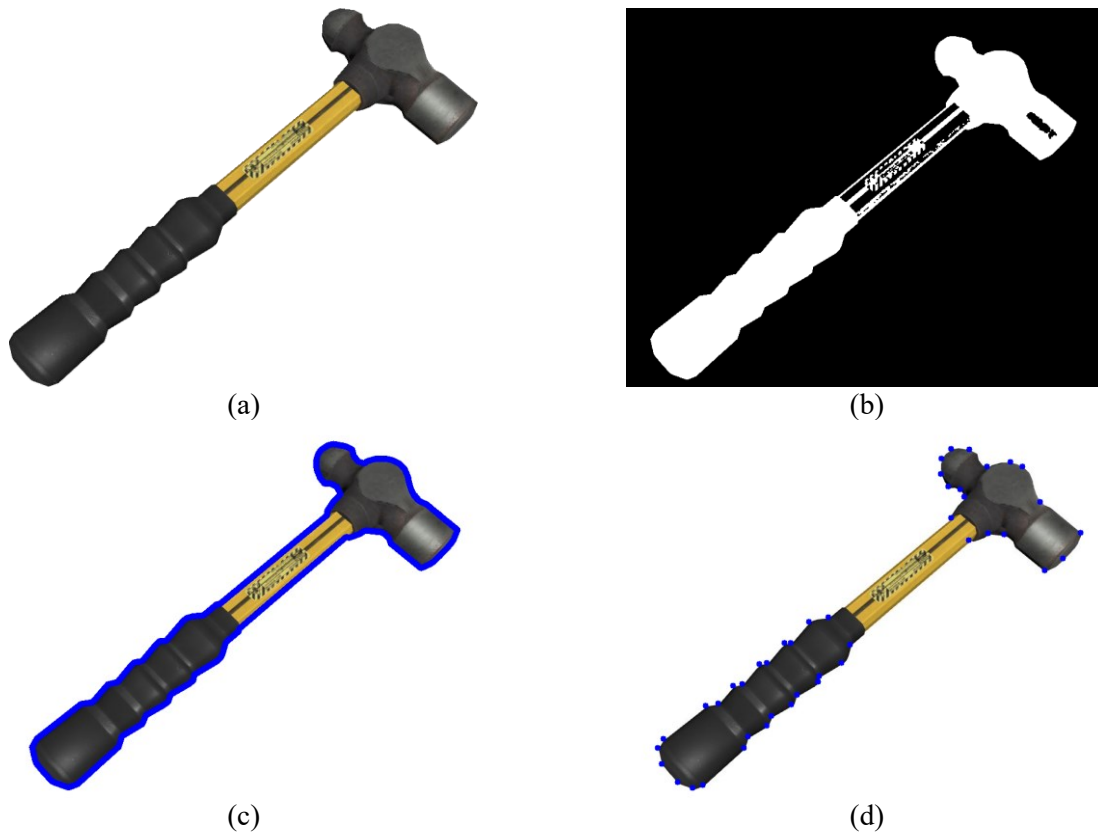
CREATE-BINARY-MASK ( $Y, K, t$ )

**in:** a grayscale image of 8-bit integer pixel intensities  $Y \in \mathbb{N}^{H \times W}$  with height  $H$ , width  $W$ , where  $Y_{i,j} \leq 255$ ; smoothing kernel  $K$ , where  $K = J_n$ , and  $J_n$  is a  $n \times n$  square matrix of ones; pixel intensity threshold  $t$ ,  $\{t \in \mathbb{N} | t \leq 255\}$

**out:** binary image mask matrix  $M \in \mathbb{N}^{H \times W}$  with height  $H$  and width  $W$ ,  $M_{i,j} \in \{0, 255\}$

```
1:   $Y \leftarrow Y || \langle \text{CONVERT-TO-GRAYSCALE}(Y) \rangle$ 
2:   $M \leftarrow \text{copy } Y$ 
3:  for all  $M_{i,j} \in M$  do
4:    if  $M_{i,j} \geq t$  then
5:       $M_{i,j} \leftarrow 0$ 
6:    else
7:       $M_{i,j} \leftarrow 255$ 
8:   $M \leftarrow \text{morph-close}(M, K)$                             ▷ apply morphological operation, erode
                                                                then dilate image (remove "holes" in
                                                                foreground object)
9:  return  $M$ 
```

---



**Figure 3.10:** Automatic polygon annotation by binary thresholding for synthetic images: (a) original image on white background, (b) object image mask from binary thresholding, (c) conversion of binary image mask to polygon points, and (d) final polygon point annotation with reduced number of points.

Using Algorithm 3.2, the binary image mask is converted to a polygon that encloses the external edges of the object. The polygon, a closed line segment, is represented as a list of x-y coordinate pairs. When the initial external contour of the object is traced with OpenCV's [77] *find-contours* algorithm, the resulting polygon had over 1000 points (Fig. 3.10c). In practice, the annotated objects usually require less than 50 points to represent the segmented object. Therefore, the object's polygon was simplified by using the Douglas-Peucker algorithm [79], which is provided in the OpenCV [77] library. The Douglas-Peucker algorithm [79] iteratively selects two points on a given line segment at opposite ends, and the points in between the sliced line segment are removed if they do not meet some tolerance value  $\epsilon$ .  $\epsilon$  in this case, represents the maximum error that the approximated polygon deviates from a given polygon. By using the Douglas-Peucker algorithm [79], the number of points required to annotate the objects in the images were consistently less than 50 points (Fig. 3.10d). Fig. 3.10 shows the conversion process of a binary image

mask to a polygon of the annotated object. The final polygon annotation of the object was manually modified in LabelMe for all objects with a reflective surface, since the white glare on the object’s edge blends with the white background of the image (Fig. 3.11).

---

**Algorithm 3.2:** Conversion of a binary mask of an object to x-y polygon points.

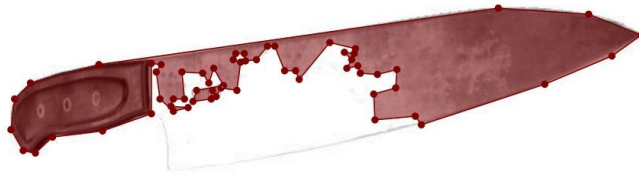
---

BINARY-MASK-TO-POINTS( $M, d$ )

**in:** binary image mask matrix  $M \in \mathbb{N}^{H \times W}$  with height  $H$  and width  $W$ ,  $M_{i,j} \in \{0,255\}$ ; deviation factor  $d$  for point reduction.

**out:** optimal contour of the segmented foreground object as a polygon object: a 2D vector of x-y coordinate pairs  $P^* \in \mathbb{R}^{N \times 2}$  with  $N$  points.

- 1:  $L \leftarrow \text{find-contours}(M)$  ▷ find binary mask edges and compute line segment as x-y coordinate pairs
  - 2:  $P \leftarrow \max S$  ▷ save line segment/contour with the most x-y points,  $P$  is the closed line segment of the polygon
  - 3:  $\epsilon \leftarrow d \cdot \text{arclength}(P)$  ▷ maximum deviation of approximated polygon from polygon  $P$
  - 4:  $P^* \leftarrow \text{approx-polygon}(P, \epsilon)$  ▷ optimal approximation of polygon  $P$  with the reduced number of points
  - 5: **return**  $P^*$
- 



**Figure 3.11:** Example of an object with an inaccurate polygon annotation (displayed in LabelMe [76]) after using Algorithm 3.2, due to the indistinctive edges from the natural sheen on the knife’s blade.

Once the real and synthetic images were all annotated for the object polygons and binary image masks, the objects were cropped from the background by subtracting the pixels contained in the object binary mask. The new background-less images were then converted to RGB-A images, which are RGB images with the transparency alpha channel, so that the objects can be overlaid on a random background during synthetic image generation. The program used to convert the images from RGB to RGB-A is a modification of the open-source repository by Gilbert Tanner [79]. The original program [79] discarded the LabelMe JSON files, which meant that binary thresholding would need to be applied again on the

single-object images during data synthesis. As mentioned previously, binary thresholding only works well if there is a visible distinction between the background and an object's edge. Instead, each RGB-A single-object image retains the corresponding LabelMe JSON annotation file of an object's class and the object's polygon in x-y coordinates. The top-left corner of the cropped object's bounding box was also aligned to the top-left corner of the new RGB-A image by subtracting each x-y coordinate of the object's polygon with the top-left x-y coordinate of the original object's bounding box.

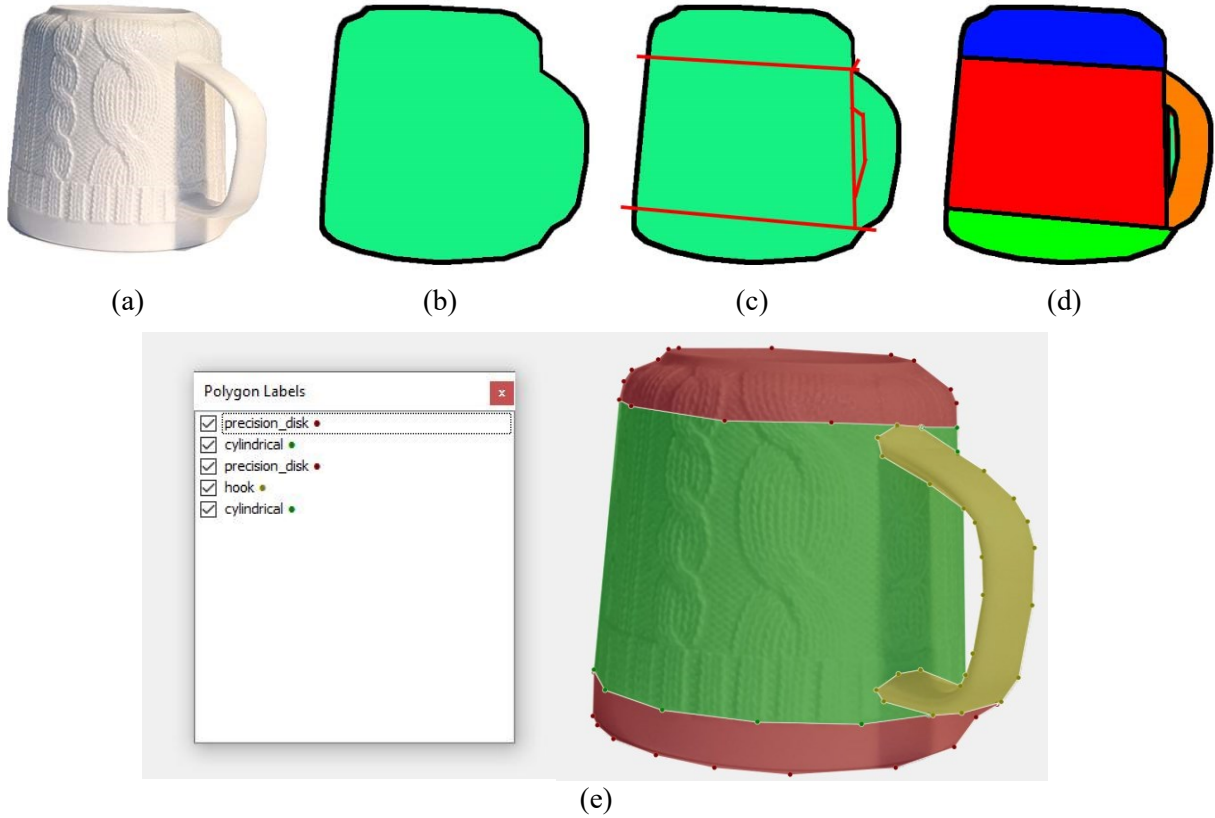
### 3.2.4 Annotation of Grasp Affordances as Instances of Image Masks

Given that a polygon object annotation exists for each single-object RGB-A image, the grasp affordances were annotated by partitioning the object's polygon into smaller polygons of a unique object part. Each of the object parts were directly classified as a grasp affordance, instead of explicitly classifying each object part by its name (e.g., a bottle cap). As mentioned at the end of Sec. 3.1.3, the grasp affordances are represented as grasp-type regions and task regions. The grasp types are directly labelled on each object part, meaning that a single object part corresponds to a single grasp-type class. However, since the same task can be performed on various parts of an object, the task regions are annotated by combining the polygons of the object parts. For example, to *use* a mug, the handle and the body of the mug can be grasped. Therefore, the handle and the body of the mug are labelled with the *use* task. The annotation method of combining the polygons from the object parts also accounts for the overlapping regions between the different task classes. For example, a mug can be *carried* and *used* by grasping the handle; therefore, the handle can be labelled with the *carry* and *use* task. Note that the *carry* task is exclusively labelled for the regions on an object that provides a safe and stable grasp location to transport with the object (e.g., walking with the object). Whereas, the *use* task includes all other object class-specific tasks, such as, to *pour*, in the case of a mug.

#### 3.2.4.1 Annotation of Grasp-Type Regions

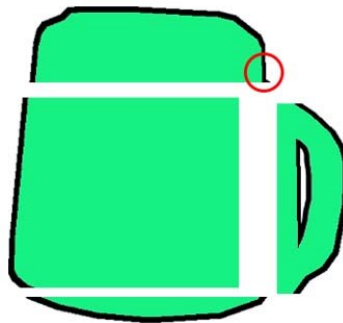
Before the object parts were labelled as grasp types, a polygon partitioning tool, that follows Algorithm 3.3, was developed to divide an input object polygon from a given set of pre-annotated line segments and polygons. For each RGB-A image and in their respective object annotation file, the additional line segments and polygons were annotated using LabelMe. As shown in Fig. 3.12c, the line segments indicate a dividing line in the object's polygon. By using Algorithm 3.3, the annotation process was simplified, since the line segments automatically split the object mask into smaller polygons, to eliminate the need to manually annotate a new polygon for each object part. Occasionally, an object part was

positioned far from the edges of the object’s polygon and the dividing lines. Therefore, the nested object parts were directly annotated as another polygon. In addition, the pre-annotated polygons enabled the annotations of a hole within the object’s polygon, where the annotations at the hole should be omitted, since the region was part of the image background (Fig. 3.12c). The holes in the object’s polygons were reconverted to a binary image mask, and the image pixels within the hole were deleted from the image (Fig. 3.12e). Unfortunately, the polygons in LabelMe could only annotate the external edges of a polygon. When the polygons were later re-converted to binary image masks, only the background pixels could be physically removed from the image. Unless the hole in the object’s polygon coincides with the edges of the object or an object part (Figs. 3.12c and 3.12e), then the binary image masks of the object or object part would contain the annotations of the hole.



**Figure 3.12:** Polygon split for the grasp-type annotations from Shapely [81] to LabelMe [76]: (a) original image of a cup, (b) cup’s contour and image mask, (c) object mask with the dividing lines, (d) divided polygon regions, and (e) the annotated grasp-types on the polygons with the removed hole adjacent to the cup’s handle.

In Algorithm 3.3, the functions used to manipulate the polygons and the line segments were implemented using the Shapely library [81]. The pre-annotated polygons of object parts or holes  $P_U$ , were first subtracted from the object's polygon  $P_O$ . The line segments of the resulting difference polygon of  $P_O$  were merged with the pre-annotated line segments  $L$ . The *polygonize* command in Shapely then automatically returned all the divided polygons from the merged line segments  $L_M$ . To eliminate the artifacts of small polygons from the divided polygons  $P_M$ , the area of each divided polygons  $P_{M_i}$  were measured in relation to the difference polygon  $P_O$  by calculating a ratio between  $P_{M_i}$  and  $P_O$ . The polygon was only included in the final polygon annotation file if the area ratio between  $P_{M_i}$  and  $P_O$  was more than 0.001. As seen in Fig. 3.13, the accepted polygon may contain residual artifacts along the edges. Similar to Sec. 3.2.3, the small fragments along the edges of the polygon were filtered with a morphological closing transformation, by calculating the Minkowski difference, followed by a Minkowski sum, between the polygon and a small circle with a radius of 0.5 pixels. Also similar to Sec. 3.2.3, the contour of the polygon was again simplified with the Douglas-Peucker algorithm [79] to reduce the number of x-y points required to form the polygon. All of the filtered polygons  $P_S$ , were then saved into a new LabelMe JSON file without the original object's polygon.



**Figure 3.13:** Example scenario where a residual *thin rectangle* (circled in red) exists after dividing the object image mask into smaller polygon regions.

Finally, for every image, each polygon annotation of an object part was manually relabeled with a grasp type that matched the shape and the type of task that can be accomplished with the object part. The object parts could be labelled with one of the following grasp types: adducted thumb, cylindrical wrap, hook, lateral tripod, light tool, precision disk, and prismatic 4 finger. The illustrations of each of the listed grasp types are shown in Table 2.1 in Sec. 2.2.

---

**Algorithm 3.3:** Division of a polygon annotation of an object by given line segments and polygons.

---

SPLIT-OBJECT-POLYGON( $P_O, P_U, L, a, r$ )

**in:** x-y coordinates of the object's polygon  $P_O \in \mathbb{R}^{N \times 2}$  with  $N$  points; a set of line segments  $L$ , where each line  $L_i \in \mathbb{R}^{N \times 2}$  has  $N$  x-y coordinates; a set of pre-defined polygons  $P_U$  ( $P_{U_i} \in \mathbb{R}^{N \times 2}$ ) that can either be a hole in the object's polygon, or an existing annotation of a polygon; the minimum area tolerance value  $a$ , that a polygon is accepted based on the ratio between the polygon in question and the object's polygon; radius  $r$  of a circle object  $C_r$  for polygon simplification.

**out:** A set of divided polygons  $P_S$ , where  $P_{S_i} \in \mathbb{R}^{N \times 2}$  has  $N$  points.

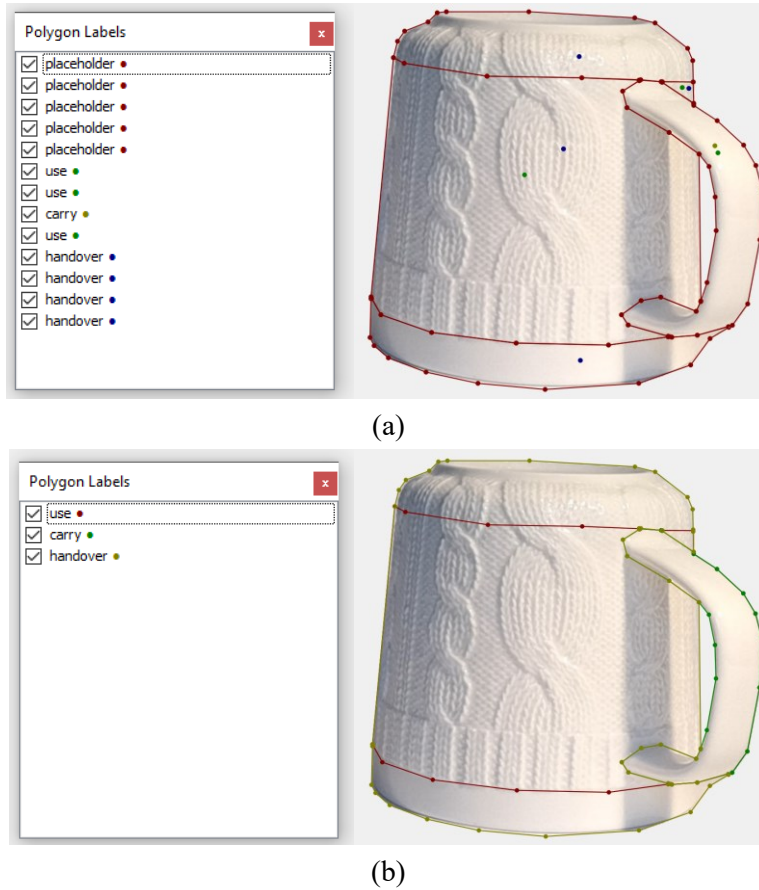
```
1:    $P_S \leftarrow \emptyset$                                 ▷ initialize an empty set of polygons (output)
2:   if  $P_U \neq \emptyset$  then                            ▷ check if there are existing annotations of holes
                                                and polygons
3:       for all  $P_{U_i} \in P_U$  do
4:            $P_O \leftarrow P_O - P_{U_i}$                 ▷ Subtract the area of  $P_O$  and  $P_{U_i}$ , update  $P_O$  (now
                                                the difference polygon)
5:           if  $\text{label}(P_{U_i}) \neq \text{hole}$  then
6:                $P_S \leftarrow P_S \cup \{P_{U_i}\}$     ▷ append polygon to  $P_S$  if it is not a hole
7:            $L_M \leftarrow \text{merge}(\{L_O \in P_O\} \cup L)$   ▷ merge all lines with all lines that form the
                                                object's polygon
8:            $P_M \leftarrow \text{polygonize}(L_M)$           ▷ convert the merged lines and divide into
                                                smaller polygons
9:       for all  $P_{M_i} \in P_M$  do                    ▷ iterate and filter the divided polygons
10:           $a_p = \text{area}(P_{M_i}) \text{ div } \text{area}(P_O)$     ▷ calculate the area ratio between current polygon
                                                and difference polygon  $P_O$ 
11:          if  $a_p \geq a$  then
12:               $P_{M_i} \leftarrow (P_{M_i} \ominus C_r) \oplus C_r$   ▷ Minkowski difference and sum on a circle
                                                object,  $C_r$ , with radius  $r$  (morphological closing)
13:               $P_{M_i}^* \leftarrow \text{approx-polygon}(P_{M_i}, r)$   ▷ optimal approximation of polygon  $P$  with the
                                                reduced number of points
14:               $P_S \leftarrow P_S \cup \{P_{M_i}^*\}$ 
15:   return  $P_S$ 
```

---



### 3.2.4.2 Annotation of the Task Regions for Task Suitability

With the resulting polygon annotation files from Sec. 3.2.4.1, the task regions were automatically annotated using Algorithm 3.4, provided that a set of x-y point annotations and their respective task labels were given; the functions from the Shapely library was used in Algorithm 3.4 to manipulate the polygons and points. A *carry*, *use*, or *handover* task label was manually assigned to an object part, by annotating an x-y point inside the polygon of the object part (Fig. 3.14a). Each object part can be annotated with multiple x-y points from different task classes. As mentioned at the beginning of Sec. 3.2.4, the *carry* task is dedicated to the object parts that can be most stably grasped to transport the object, while the *use* task is for other object class-specific tasks. A *handover* task constitutes all object parts that allow the user to safely pass the object to another hand without obstructing the primary location to carry the object (e.g., a handle). In effect, the suitable task regions account for user safety and the accessibility of the grasping location, unlike the annotations of grasp types, which primarily only considered the object part's shape. For this reason, the original polygon object annotation of all knives was further divided to include the bolster (the area where the handle meets the blade) and the cutting edge as a unique knife part, using the same method proposed in Sec. 3.2.4.1. The bolster of a knife can be used to handover a knife to leave the handle open for grasping, while the cutting edge of a knife is discarded from all task-region annotations (no x-y points are labelled at the cutting edge) since it is hazardous to the user. Once the x-y points of the task regions were annotated, Algorithm 3.4 was implemented to automatically label the polygon regions of the object parts with the corresponding tasks. The labelled polygons were also combined with the adjacent polygons that were labelled with the same task (Fig. 3.14b). Therefore, no further annotation or relabeling of the object parts was required after using Algorithm 3.4.



**Figure 3.14:** Point annotation process in LabelMe [75] for the task regions: (a) point locations of each task, where the *placeholder* regions are the unlabeled polygon regions, and (b) combined polygon regions by task class after using Algorithm 3.4.

In Algorithm 3.4, the given point annotations in an image are first matched with the locations of the polygons of the object parts. If a point lies within the polygon, then the polygon is appended to the set of polygons with the matching label. For example, if the annotated point is labelled with the *carry* task, then the selected polygon is also labeled with the *carry* task. At this stage, the polygons are only included into each corresponding set of polygons with their corresponding task labels, unmerged. Once all of the given annotated points are matched with the polygons, any polygon that remains without a corresponding labeled point annotation will be discarded from the final annotation file. If a point lies on multiple polygons, then the polygon with the smallest area will be selected. The polygons in their respective task classes are then merged (e.g., all polygons labeled with the *carry* task are internally combined separately

from other polygons with a different task label). All polygons belonging to the same task class that are not coincident with each other are left unmerged as a separate entry in the final annotation file.

---

**Algorithm 3.4:** Combining polygons from a set of given points and labels for automated task-region annotations.

---

COMBINE-POLYGONS( $P_S, Q, l_d, l_e, t$ )

**in:** x-y coordinates of a set of unlabeled coinciding polygons  $P_S$ , where  $P_{S_i} \in \mathbb{R}^{N \times 2}$  has  $N$  points; a set of x-y points  $Q$  with corresponding class labels  $L_Q \in \mathbb{N}^{|Q|}$ , where each point  $Q_i \in \mathbb{R}^2$  has a corresponding class label  $l_{Q_i} \in \{0,1,2\}$ ; the default class label  $l_d \in \{0,1,2\}$ ; the exclusion class label  $l_e \in \{0,1,2\}$ , for the labelled polygons omitted  $P_{l_e} \in P$  from the default polygon  $P_{l_d} \in P$  with label  $l_d$ ; tolerance  $t \in \mathbb{R}$  for the maximum distance between two adjacent polygons.  
**out:** A set of organized, class-based polygons  $P = \{P_0, P_1, P_2\}$ , where  $P_l$  contains all polygons with the corresponding class labels  $l \in \{0,1,2\}$ , and  $P_{l_i} \in \mathbb{R}^{N \times 2}$  has  $N$  points.

```

1:    $P \leftarrow \emptyset$                                 ▷ initialize an empty set of polygons (output)
2:   for all  $Q_i \in Q$  do
       $J \leftarrow \text{indices}(Q_i \in P_S)$            ▷ return the indices where a given point is
                                                    within any of the polygons  $P_S$ 
3:     if  $|J| = 1$  then                               ▷ check for a single matching index
4:        $j \leftarrow J$                                ▷  $J$  contains only one index  $j$ 
5:     else if  $|J| > 1$  then                          ▷ check for nested polygons
6:        $j \leftarrow \arg \min(\text{area}(P_S[J]))$         ▷ index  $j \in J$ , find the polygon with the
                                                    smallest area
7:      $P_{l=\text{label}(P_S[j])} \leftarrow P_S[j]$         ▷ save the polygon to one of the subsets of  $P$ 
                                                    with the matching label
8:     if  $l_d \neq \text{NIL}$  then                          ▷ default polygons (if exists) are all polygons
                                                    minus the excluded polygons  $P_{l_e}$ 
9:     if  $l_e \neq \text{NIL}$  then
10:       $P_{l_d} \leftarrow P \setminus \{P_{l_e}\}$ 
11:     else
12:       $P_{l_d} \leftarrow P$ 
13:     for all  $P_l \in P$  do
14:       $P_l \leftarrow \text{merge}(P_l)$                  ▷ initial merge of each subset of polygons for
                                                    each class label
15:     if  $|P_l| > 1$  then
16:       $P_l \leftarrow \text{explode}(P_l)$                  ▷ unmerge polygons
17:       $C \leftarrow \{\{X, Y\} | X, Y \in P_l \wedge X \neq Y\}$ 
                                                    ▷ find all pair-wise combinations, without
                                                    repetition, of the smaller polygons of label  $l$ .
18:      $P_l \leftarrow \emptyset$ 
19:     for all  $C_i \in C$  do
20:       if  $|\text{distance}(C_{i_1}, C_{i_2})| < t$  then    ▷ absolute distance between pairwise
                                                    polygons
21:          $P_l \leftarrow P_l \cup \text{merge}(C_i)$       ▷ append the merged polygons
22:   return  $P$ 

```

---

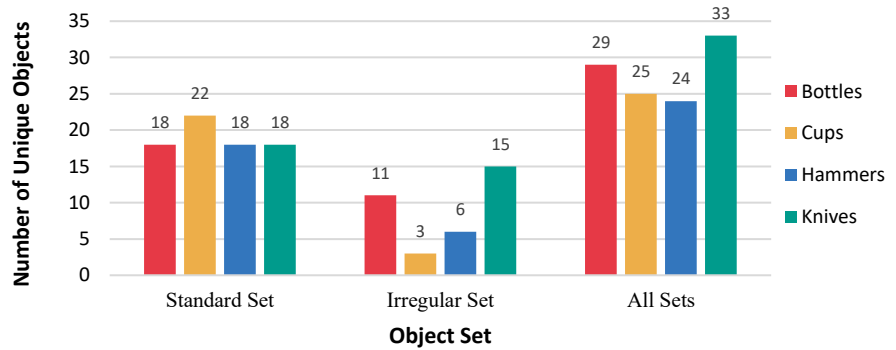
### 3.2.5 Organization of the Foreground Images and Annotations of Captured Objects into Standard and Irregular Object Sets

The annotated single-object RGB-A images were used as the foreground images to generate the MOMA synthetic dataset on random background images. A total of 111 objects produced 2777 foreground images, where the corresponding images and annotations of each object were included in either standard or irregular object sets. The captured objects were divided into the two object sets, since the grasp-type and task regions change when the object's shape and features deviate from the objects of the same object class. Therefore, the categorization of the objects, as standard or irregular objects, was based on the overall appearance and functionality of the objects, as shown in Table 3.2.

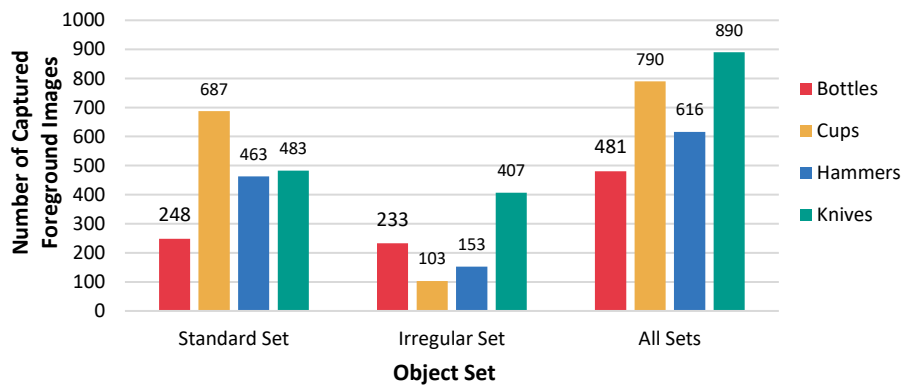
**Table 3.2:** Evaluation criteria to classify and categorize each unique object, by object class and object set, respectively.

Object Class	General Definition	Additional Criteria for Each Object Set	
		Standard Object Set	Irregular Object Set
Bottle	Any tall, narrow-necked, and lidded container that can hold liquids.	The object is axially symmetrical when observed from the top.	The cap of the bottle can be a dispensing medium (such as a spray bottle), and the object does not necessarily have to be radially symmetrical when viewed from the top.
Cup	Any short container that generally has a wide-open top. The object can hold liquids and may have a handle.	Like the bottle, the body of the object is axially symmetrical when observed from the top. The body of the object is cylindrical with a uniform diameter.	The body of the object does not need to have a uniform diameter (such as a wine glass).
Hammer	Any long tool with a long handle and a blunt head, for tasks such as driving nails.	The tool head is metal, and roughly has the same diameter as the handle.	The tool head can be non-metal, and its diameter can vary from the handle.
Knife	Any sharp tool with a handle and a flat blade for tasks such as cutting and paring.	The width of the blade and handle are relatively constant (no blade heel).	The blade of the object can be significantly larger than the handle in terms of width. The handle of the object features a blade guard.

After using Table 3.2 to categorize the objects into the standard and irregular object sets, there were 76 objects in the standard object set and 35 objects in the irregular object set. Fig. 3.15 summarizes the number of unique foreground images and objects in both standard and irregular object sets. There is a class imbalance in the distribution of the objects and foreground, by object class, even when the number of objects in each object class are mostly equal, as in the case of the standard object set. As described in Sec. 3.2.1, the class imbalance is because a varied number of images of unique poses are captured for each object. However, the object class imbalance issue is eliminated during data synthesis, since there is an equal probability that an object from each object class is placed on the synthetic images.



(a)



(b)

**Figure 3.15:** Distribution of all objects across all object classes and object sets by: (a) number of objects and (b) number of captured foreground images.

### 3.3 Creation of the MOMA Synthetic Dataset

The MOMA synthetic dataset contained 20K images that are divided into 5 unique datasets for training, validation, and testing. Each synthetic image was generated by randomly selecting and placing foreground images over a random background image (Fig. 3.16). The foreground images are the annotated single-object RGB-A images from Sec. 3.2, where each RGB-A image has a single-object on a transparent background. Every single-object RGB-A image has three individual LabelMe JSON annotation files of the object, grasp-type regions, and task regions, where each annotation contains the x-y coordinates of a polygon and the class label from one of the annotation categories. The code used to synthesize the images for the MOMA dataset was based on the implementation of the synthetic dataset generator from [80].



**Figure 3.16:** Sample synthetic image generated for the MOMA dataset, containing the foreground objects of a cup, hammer, and bottle.

#### 3.3.1 Development of Synthetic Dataset Generation Tool and Procedure for Generating Synthetic Images in COCO JSON and YOLO TXT.

The goal of the synthetic dataset generator is to simultaneously automate image generation and the annotation process. The original annotations of each foreground image are transferred to the synthetic image, so that every generated synthetic image has the transformed annotations of all foreground images across all annotation categories (objects, grasp types, tasks).

To generate a synthetic image, a random number of foreground images and a random background image is selected and resized. The foreground images are scaled-down based on the resulting user-specified output image dimension of the background image (the statistics of the MOMA dataset is

discussed in Sec. 3.3.2). Each foreground image is resized and relocated, by applying the following affine transformation function in homogenous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (3.6)$$

where  $x'$  and  $y'$  are the transformed coordinates of all pixels in the foreground image from the original position at  $x$  and  $y$ ,  $s_x$  and  $s_y$  are the scaling factors in the  $x$  and  $y$  directions, respectively; and  $t_x$  and  $t_y$  are the translation of the foreground image's  $x$  and  $y$  coordinates, respectively. Essentially,  $t_x$  and  $t_y$  are the new top-left coordinates of the transformed foreground image in the composite image since the top-left position of the foreground image is aligned to  $(0, 0)$ , prior to the affine transformation in Eq. 3.6. In addition, a constant scaling factor is used to resize the foreground image in both  $x$  and  $y$  directions to maintain the same aspect ratio of the original foreground image. Therefore, Eq. 3.6 can be simplified as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (3.7)$$

Using Eq. 3.7, the coordinates of all polygon annotations across all annotation categories are transformed in the same manner. The modified annotations of all foreground images are added to a new set of annotation files for each synthetic image, with one LabelMe JSON file per annotation category (objects, grasp-type regions, task regions).

Once the images are synthesized, the annotations of the synthetic images need to be converted to a COCO JSON or YOLO JSON format before the dataset can be used for model training, validation, and testing. By using the LabelMe JSON to COCO JSON formatting tool by LabelMe [76], a single, unique annotation file is generated for each annotation category, where three annotation files in COCO format are generated for each training, validation, and testing sets. Similar to Sec. 3.2.3, the annotations in COCO format contains the metadata of all images in that particular dataset, as well as the  $x$ - $y$  coordinates of the polygons and bounding boxes with its corresponding class label. For YOLO TXT format, similar to LabelMe JSON format, each synthetic image also has a corresponding annotation file; in this case, each synthetic image has three annotation files in YOLO TXT format, one for each annotation category. Each polygon annotation is converted to a one-line entry in the text file:  $[c, \hat{x}_1, \hat{y}_1, \dots, \hat{x}_n, \hat{y}_n]$ , where  $c$  is the class label as an integer value (e.g., the value 2 to represent the object class *hammer*), and each  $(\hat{x}, \hat{y})$  coordinate pair represents the normalized coordinates of each corresponding coordinates of the original

polygon coordinates. To normalize each x-y coordinate pair from a polygon annotation, the following equations were used:

$$\hat{x} = \frac{x}{w}, \hat{y} = \frac{y}{h} \quad 0 \leq \hat{x}, \hat{y} \leq 1 \quad (3.8)$$

where all x coordinates are divided by the image width  $w$ , and all y coordinates were divided by the image height  $h$ ;  $\hat{x}$  and  $\hat{y}$  are the resulting normalized coordinates for all x-y coordinates of the polygon annotation. Each of the normalized x-y coordinates can have a value between 0 and 1. Unlike COCO JSON, annotations in YOLO TXT format do not contain the metadata of the corresponding image, nor do they contain the bounding boxes of the transformed polygon annotation. However, a bounding box can be automatically created by finding the minimum and maximum values of the normalized x-y coordinates.

### 3.3.2 MOMA Synthetic Dataset Summary

Before the MOMA synthetic dataset is generated using the method developed in Sec. 3.3.1, the foreground and background images are first distributed to each of the training, validation, and test datasets. The images of the MOMA synthetic dataset are then generated accordingly to the specified output image dimensions, foreground image scaling, and the number of foreground objects in Table 3.3; the number of foreground images, unique objects, and background images in each dataset are also summarized in Table 3.4. Overall, the MOMA dataset has one training set (10K images), two validation sets (2.5K images each), and two test sets (2.5K images each). The order of the datasets in Table 3.4 represents the decreasing familiarity of the objects that a trained model is exposed to. The training, validation, and standard test datasets (Test S) all used objects from the standard object set, while the irregular object set was only used by the irregular test set (Test I). Therefore, the irregular object set was only used to evaluate a model’s performance on unconventional and unseen objects. The MOMA dataset was generated on 200 background images that were sourced from Unsplash and Pexels, which are open-source image repositories. From Fig. 3.17, the background images were a collection of real RGB images from a kitchen, dining room, office, bathroom, landscape, plain texture, and random objects.

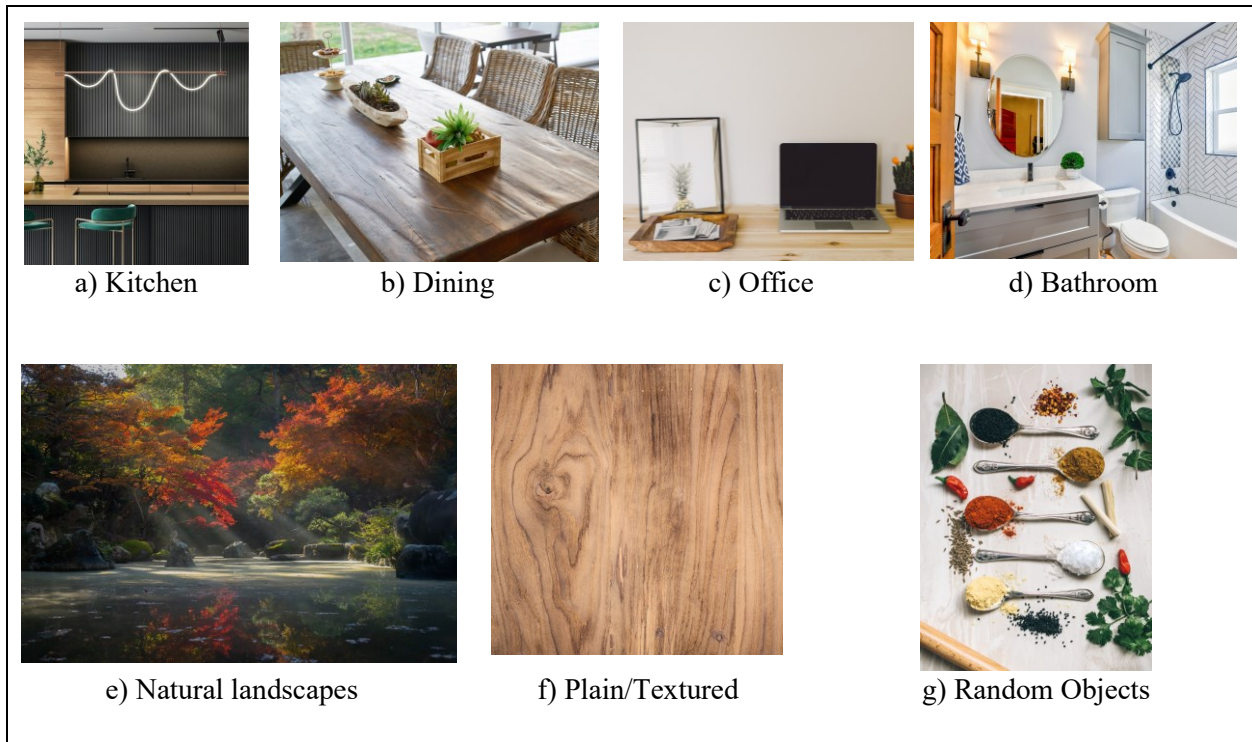


**Table 3.3:** Specifications used to generate each dataset in the MOMA synthetic dataset. All validation and test sets were generated in the same manner.

Specification	Specification Group			
	1	2	3	4
Number of objects	5-10	5-10	2-5	1-3
Object Scale (w.r.t. the image dimensions of the foreground image)	0.5-1	0.25-1	0.25-0.5	0.2-0.3
Output image dimension	1920 × 1080	1920 × 960	960 × 540	480 × 540, 540 × 480
<b>Number of images generated according to each specification for each dataset</b>				
Train Set	2500	2500	2500	2500
Validation/Test Set	1250	-	625	625

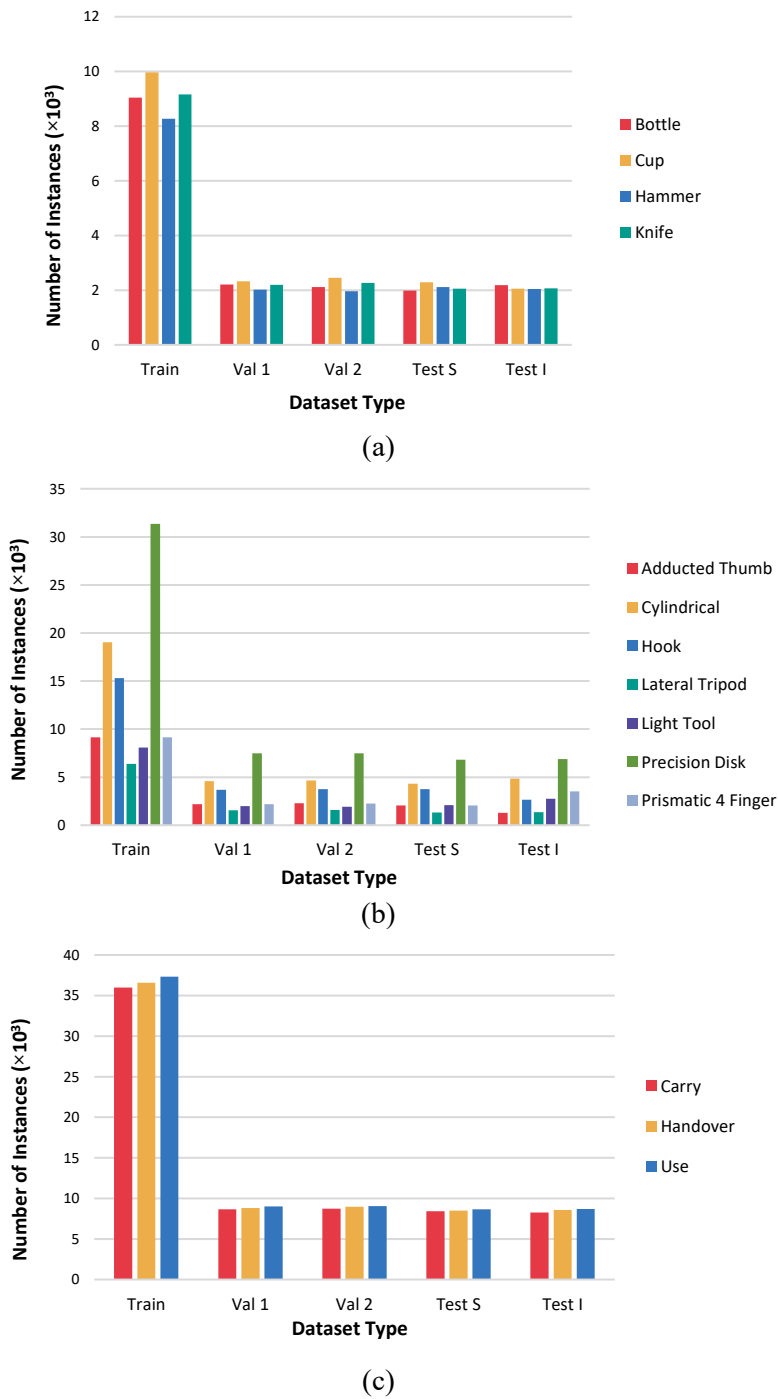
**Table 3.4:** Training, validation, and test split of the single-object RGB-A foreground images, and distribution of unique background images in each dataset.

Dataset Type and Name	Dataset Description	Number of Unique Objects per Object Class	Number of Unique Foreground Images per Object Class	Number of Unique Background Images
Train (Train-S)	Objects and foreground images from the standard set. Represents everyday common objects in common scenes.	14 bottles, 14 cups, 13 hammers, 14 knives.	152 bottles, 320 cups, 278 hammers, 277 knives.	100
Validation 1 (Val-S1)	Same objects and foreground images as the training set. New background images that differ from the training set. Represents everyday common objects in unfamiliar scenes.	<i>Same objects as the training set.</i>	<i>Same foreground images as the training set.</i>	50
Validation 2 (Val-S2)	Unique poses of the same objects (unique poses of the everyday objects) used in the training set. Same background images as Validation 1.	<i>Same objects as the training set.</i>	45 bottles, 113 cups, 84 hammers, 100 knives.	<i>Same background images as Validation 1.</i>
Test S (Test-S)	Unique objects from the standard set that are not used in the training and validation sets. New background images that differ from the training and validation sets. Represents objects similar, yet different, to everyday objects in new scenes.	4 bottles, 8 cups, 5 hammers, 4 knives.	51 bottles, 254 cups, 101 hammers, 106 knives.	50
Test I (Test-I)	All objects from the irregular set. Same background images as Test S. Represents unseen and uncommon objects in unfamiliar scenes.	11 bottles, 3 cups, 6 hammers, 15 knives.	233 bottles, 103 cups, 153 hammers, 407 knives.	<i>Same background images as Test S.</i>



**Figure 3.17:** Sample background images of used to generate the MOMA synthetic dataset.

The annotations of the completed MOMA synthetic dataset had over 70K instances of objects, 190K instances of the grasp-type regions, and 214K instances of the task regions across the five training, validation and test sets. While Fig. 3.18 shows that there is a class balance of the number of the objects and the task regions, the distribution of grasp-types are imbalanced. Some grasp types are more common than others, and a grasp type can have more than one instance on an object.



**Figure 3.18:** Instance distribution of the number of (a) objects, (b) grasp types, and (c) task regions in the MOMA synthetic dataset.

### 3.4 Object and Affordance Detection Model Architectures

The Mask R-CNN [33], YOLACT [82], and YOLOv5s-Seg [83] instance segmentation models were selected to be trained and evaluated on the MOMA synthetic dataset. All models were trained and evaluated using the PyTorch library [68]. YOLACT [82] and YOLOv5s-Seg [83] models were developed for real-time performance, while Mask R-CNN [33] prioritized overall model accuracy. Since the focus of this thesis research was to develop a Multi-Affordance Detection Network (MAD-Net) that emphasizes model accuracy, the Mask R-CNN [33] network was selected as the base model. The three-prediction head variant of the Multi-Head Mask R-CNN [33] model described in Sec. 3.4.4 was the selected model for MAD-Net. MAD-Net was therefore a joint detection network capable of simultaneously detecting the instances of objects and their grasp affordances (as grasp types, and task regions).

By using transfer learning, all the models analyzed in this section were already pre-trained on the COCO 2017 instance segmentation dataset [65]. All models tested in this section were suitable for objects in cluttered scenes, and simultaneously detected object classes, bounding boxes, and binary image masks for each detection category (objects, grasp types, and task regions).

In the proceeding sections, when the *object* was mentioned in the context of object classification, the *object* can refer to the physical objects in an image (such as a cup), or the object parts that a grasp affordance is associated with. Otherwise, the type of *object* is dependent on the specific type of detection mentioned (physical objects, grasp types, and tasks).

#### 3.4.1 Mask R-CNN

Mask R-CNN [33] (Fig. 3.20) is the Faster R-CNN [56] object detection model with an additional image mask prediction module. In contrast to Faster R-CNN, Mask R-CNN replaces the ROI Pool module with the ROI Align module. ROI Align and ROI Pool both pool the values in a feature map enclosed by the ROIs (the localized bounding boxes) predicted in the RPN. However, ROI Align is also location dependent since the coordinates of the ROI are bilinearly interpolated with the enclosed feature map values [33]. ROIAlign in instance segmentation was necessary since the boundaries of the ROI may not align with the grid of the feature maps. The accuracy of binary image mask prediction was dependent on how precise the ROI captured a potential object [33].

Initially, the input images were resized and normalized using the provided Transform module in the Mask R-CNN [33] model. The ground truth annotations (bounding boxes and corresponding binary image

masks) were also resized in the same manner as the input images, but not normalized. Using the same image normalization method in Sec. 3.1.2, the input images were normalized with the mean and standard deviation values from the ImageNet dataset [67]. Due to the GPU memory constraints, the input images were resized to maintain a minimum and maximum of 800 and 1344 pixels, respectively (the largest image in the MOMA dataset is  $1920 \times 1080$  pixels). The lengths and widths of the input images must be divisible by 32 pixels for the Mask R-CNN [33] model to run. Although the images were resized, the original aspect ratios of the images were maintained. Unlike the YOLOv5s-Seg [83] and YOLACT [82] models, however, the input images for the Mask R-CNN [33] network can have varying image dimensions and aspect ratios. For example, one image (after resizing) can have the dimensions of  $1344 \times 768$  pixels (approximately 16:9), while another image in the same batch can be  $960 \times 960$  pixels (1:1).

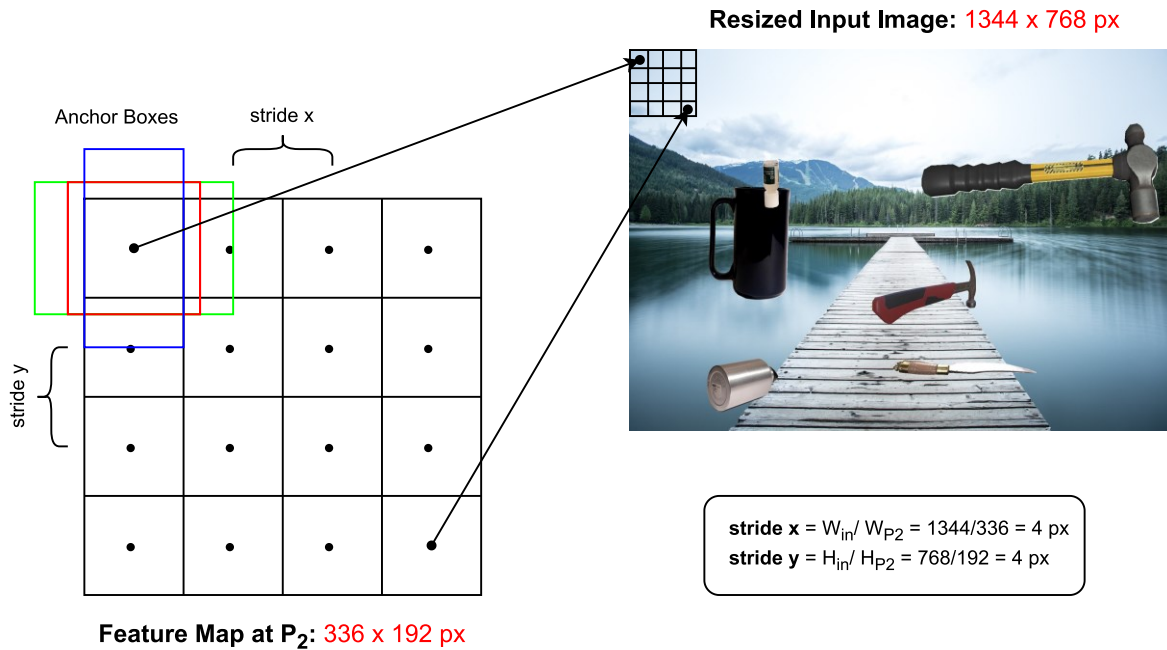
Once the input images are normalized and resized, the input images are fed into the ResNet-50-FPN backbone (Fig. 3.21) for feature extraction. From  $C_2$  to  $C_5$  in ResNet-50 [66], the outputs of each succeeding convolutional blocks are concatenated, by summation, with a convolutional block in the Feature Pyramid Network (FPN) [84] of the same feature map dimensions. For example, the output of  $C_2$  from the ResNet-50 network is connected to  $P_2$  in the FPN since they both have a feature map resolution of  $192 \times 336$  pixels. Before the layers of the ResNet-50 are concatenated with the FPN, an extra convolutional layer with batch normalization and reLU activation ( $T_N$ ) downsizes the output channels of the ResNet-50 feature maps to 256 units. In addition, each succeeding FPN layer ( $P_N$ ) upsamples (using nearest neighbours) the resolution of the previous FPN outputs ( $P_{N-1}$ ) by a factor of 2. In total, five feature maps are produced with five unique resolutions: four from the FPN, and one from  $P_6$ , which is the extra max pooling layer after  $C_5$  in the ResNet-50 network. The final feature maps from the FPN (from  $P_2$  to  $P_5$ ) are also refined with a  $3 \times 3$  kernel convolutional layer ( $S_N$ ) before they are collectively fed into the RPN for ROI generation.

The feature maps produced from the ResNet-50 FPN backbone correspond to the scaled-down representations of the resized (and normalized) input image (Table 3.5). Before the bounding boxes can be located on the input image, the RPN initially generates the anchor boxes that are placed on the resized input image based on the corresponding locations on the feature maps (Fig. 3.19). The anchor boxes are the initial bounding boxes placed on the input image before they are optimized to precisely enclose the detected object. The scaling factor between the dimensions of the resized input image and the feature map

determines the stride, or the frequency of the placement of the anchor boxes on the resized input image. For example, in Fig. 3.19, since the scaling factor is 4 for a feature map produced at  $P_2$ , a set of anchor boxes are placed at regular intervals of 4 pixels in the x and y directions on the resized input image. Since all locations on a feature map correspond to the anchor box placement in the resized input image, each feature map produces  $W \times H \times 3$  anchor boxes, where  $W$  and  $H$  are the height and width of the feature map, respectively. For each set of anchor boxes, three aspect ratios are used (1:1, 1:2, 2:1). Therefore, three anchor boxes are placed in the respective locations on the resized input image, as determined by the locations in the feature maps. As seen in Table 3.5, a unique anchor size is also used for each feature map.

**Table 3.5:** Mask R-CNN [33] anchor specifications for each feature map produced at the FPN and ResNet-50 models. The stride values are rounded up to the nearest pixel.

Feature Map Level ( $P_N$ )	Feature Map Dimensions ( $W \times H$ )	Stride ( $x \times y$ )	Anchor Size	Anchor Aspect Ratios
$P_2$	$336 \times 192$	$4 \times 4$	32	1:1, 1:2, 2:1
$P_3$	$168 \times 96$	$8 \times 8$	64	
$P_4$	$84 \times 48$	$16 \times 16$	128	
$P_5$	$42 \times 24$	$32 \times 32$	256	
$P_6$	$21 \times 12$	$64 \times 64$	512	



**Figure 3.19:** Anchor generation example for an input image (after resized in Mask R-CNN) and a feature map from  $P_2$  of Mask R-CNN [33].

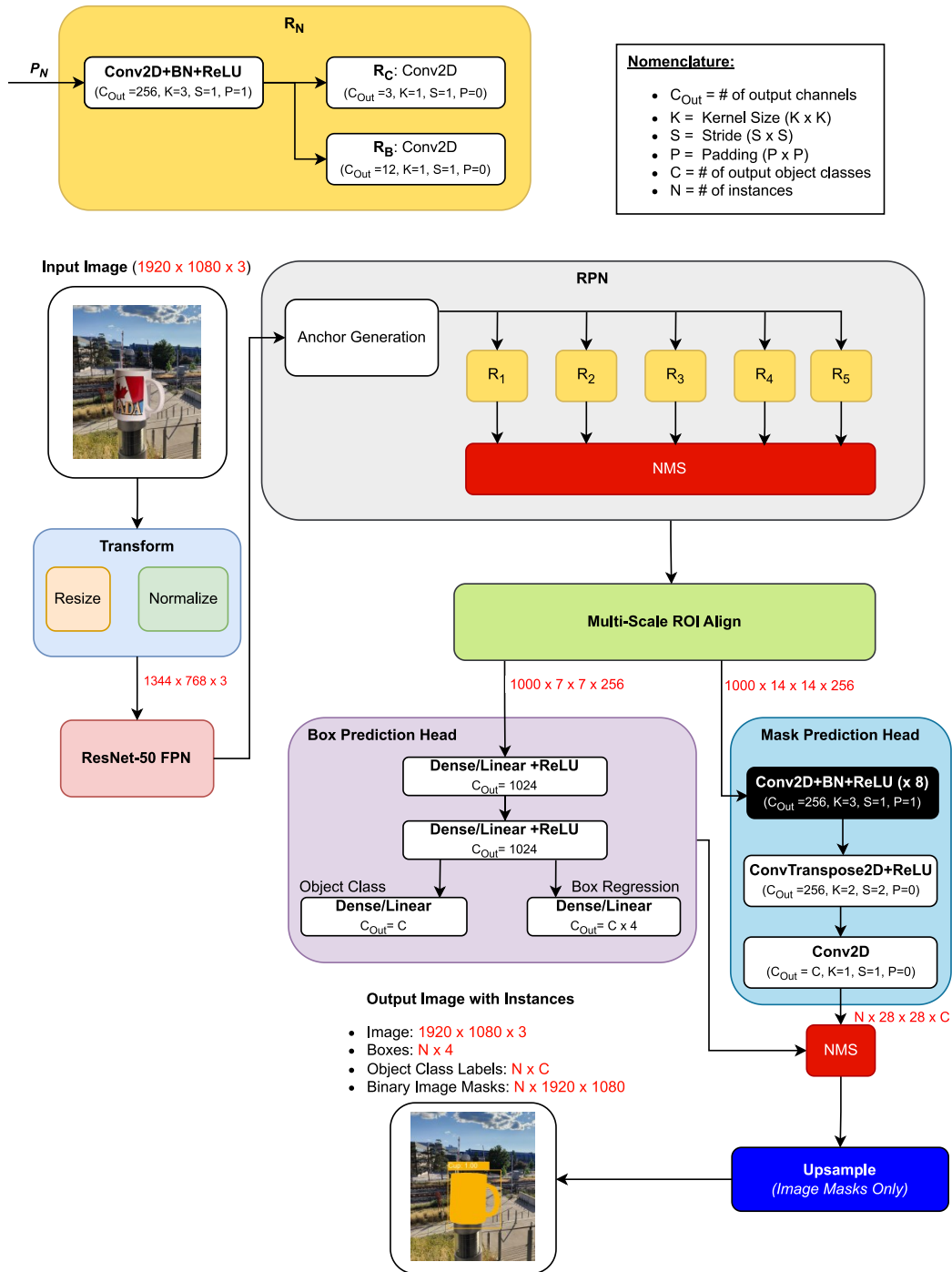
After anchor generation, the ROIs, as bounding boxes, are predicted through another CNN in the RPN. At this stage in the Mask R-CNN network, the detections of the objects are only based on the likelihood of the existence of an object (the objectness); therefore, each of the predicted bounding boxes do not have an assigned object class. In the CNN of the RPN, one convolutional layer is added for every feature map ( $R_N$ ), before the succeeding layers in the RPN predict the objectness and the change in coordinates to align the anchor boxes to a supposed ROI. Both class logits for the objectness and the bounding box predictions (the ROIs) are determined with one final convolutional layer each, with  $R_C$  and  $R_B$ , respectively. The output channel of  $R_C$  is 3, since an objectness value is predicted for each of the anchor box aspect ratios. The output channel of  $R_B$  is 4, since each bounding box is a 4-D vector of the predicted bounding box locations,  $[x_1, y_1, x_2, y_2]$ . The final ROIs are filtered with Non-maximum Suppression (NMS), to reduce the total number of ROIs to 2000 during training, and 1000 during testing. The threshold value used for NMS was 0.7, which is the mIOU value between the ROI in question and all other ROIs predicted from the same feature map. Each of the predicted ROIs is determined from one of the five feature maps. Therefore, each of the ROIs were filtered independently based on the scale of the feature map with respect to the resized input image. In addition, the different scaling of each feature map



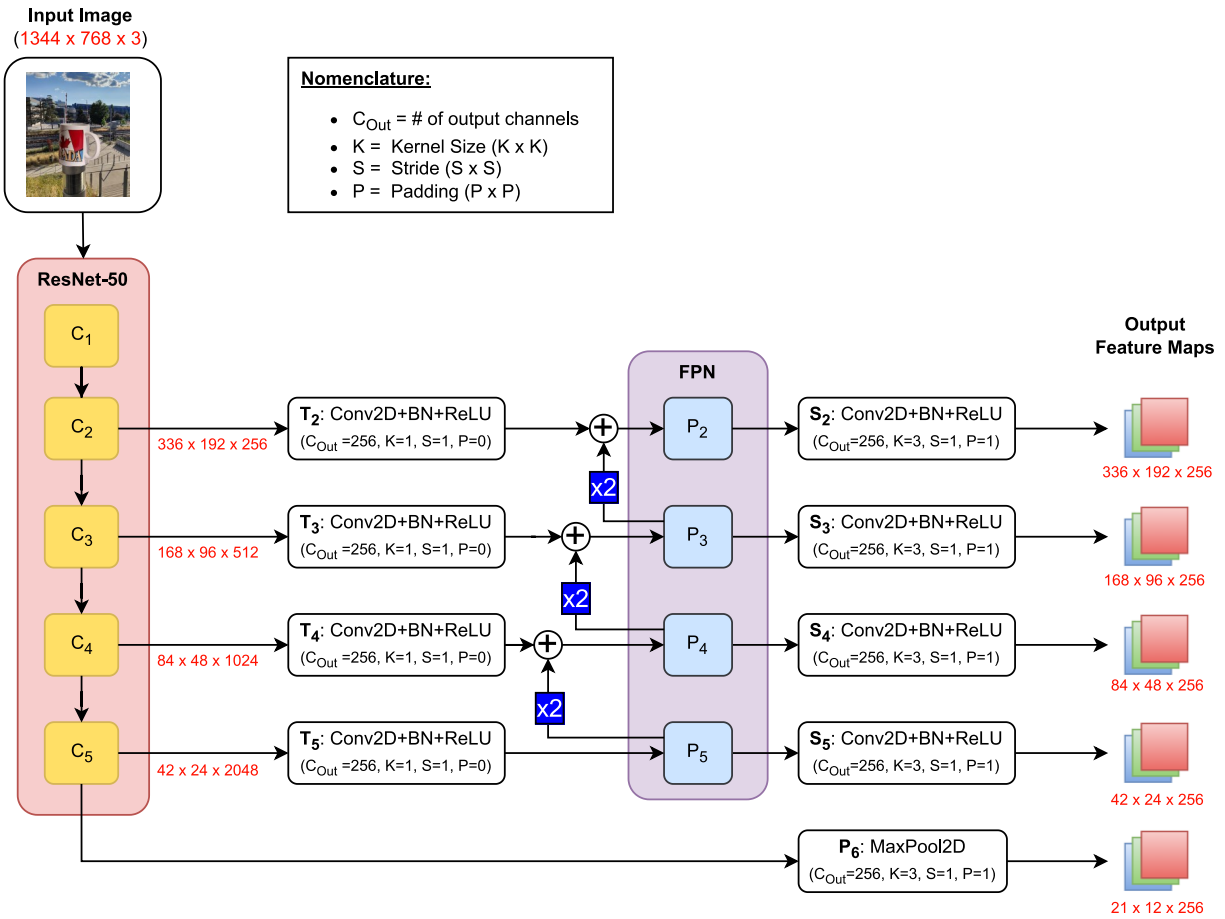
enables the RPN to detect objects of different sizes. The larger the feature map is, the higher the likelihood of detecting smaller objects, since the resolution of the feature map becomes finer. By using ROIAlign, each ROI was then used to extract a single  $7 \times 7$  feature map and aligned with the corresponding location in the resized input image.

The object's bounding boxes, class, and binary image masks were predicted in three parallel networks. The object class and bounding box network branches (box head) first flatten the  $7 \times 7$  feature maps into a 1D vector with two linear dense layers of 1024 units. These linear layers were activated with reLU. The object class was then predicted in a linear classification layer that reduces the 1024 linear units to a single logit for each object class. On the other hand, the bounding boxes were determined similarly as the object classes, except the 1024 linear units were passed in a linear regression layer to be reduced to a 1D vector. The 1D vector contains the two coordinate pairs that make up the bounding box,  $[x_1, y_1, x_2, y_2]$ .

The binary image masks were predicted in the mask head that first upsamples the  $7 \times 7$  feature maps to  $14 \times 14$  feature maps. Eight convolutional layers proceed, while maintaining the feature map sizes and the number of channels (256). To obtain the binary image masks, a transpose convolutional layer (an upsampling layer with trainable weights) doubles the size of the  $14 \times 14$  feature maps to  $28 \times 28$ . A final convolutional layer then downsizes the number of channels to match the total number of object classes before all feature maps, which become the binary image masks, are bilinearly upsampled to the original input image dimensions. NMS was performed again with a 0.7 threshold to reduce the number of detected instances (object class, bounding box, and binary image mask) to the top 100 best-scoring detections.



**Figure 3.20:** Architecture of the Mask R-CNN [33] model. All image dimensions (in red) are listed in the following order: number of proposals or instances (if applicable), width, height, and number of channels. The image dimensions vary depending on the dimensions of the input image.



**Figure 3.21:** Architecture of the ResNet-50 FPN Network. All image dimensions (in red) are listed in the following order: width, height, and number of channels. The image dimensions vary depending on the dimensions of the input image. The blue  $\times 2$  blocks indicate an upsampling layer, doubling the previous feature map size.

### 3.4.2 YOLACT

The YOLACT model (Fig. 3.22), accepts input images of all sizes. However, the received images are all resized to  $550 \times 550$  pixels. The input images are also normalized with the mean and standard deviation values from the ImageNet dataset [67]. Like Mask R-CNN [33], YOLACT uses a ResNet-50 backbone with a similar FPN structure for multi-scale feature map predictions. However, the model network connections between ResNet-50 and FPN are modified, and each convolutional layer outside of the ResNet-50 model does not use batch normalization. As seen in Fig. 3.23, the skip connections begin between the  $C_3$  and  $P_3$  layers, instead of the  $C_2$  and  $P_2$  layers as in Mask R-CNN. The connections

between each FPN layer in Mask R-CNN [33] flows in one direction, with each succeeding layer downsampling the previous layer as seen from  $P_2$  to  $P_5$ . However, in YOLACT [82], the  $P_5$  layer in the FPN branches into an upsampling layer and a downsampling layer. From  $P_5$  to  $P_7$ , the FPN layers are downsampled with a convolutional layer, whereas the FPN layers are upsampled from  $P_5$  to  $P_3$ . All the feature maps generated for bounding box, binary image mask, and object class predictions are only from the outputs from the FPN. Each of the upsampling layers in YOLACT [82] uses bilinear interpolation, instead of nearest neighbours sampling like in Mask R-CNN [33].

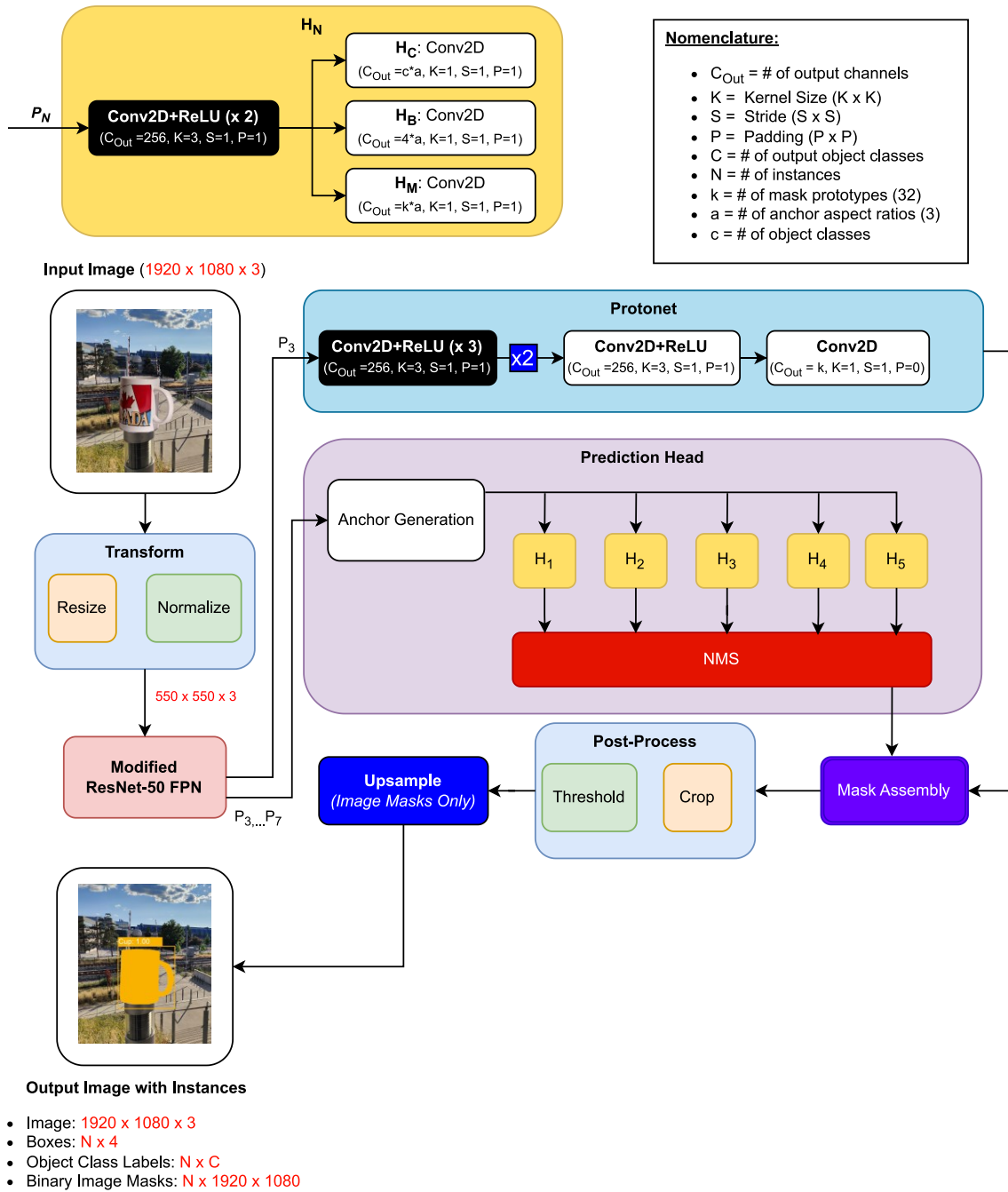
A smoothing convolutional layer ( $S_N$ ) was similarly applied to the feature maps from the FPN, before being fed into the prediction head and mask prototyping network (Protonet) [82]. Unlike Mask R-CNN [33], YOLACT [82] eliminates the need for a RPN and ROIAlign module; however, the same anchor generation method was adapted from Mask R-CNN [33], where three aspect ratios (1:1, 1:2, 2:1), and a single anchor size was used for each of the five feature maps [82]. Table 3.6 shows the expected feature map dimensions, stride on the resized input image and the anchor sizes, similar to Table 3.5 for Mask R-CNN [33].

**Table 3.6:** YOLACT [82] anchor specifications for each feature map produced at the FPN and ResNet-50 models. The stride values are rounded up to the nearest pixel.

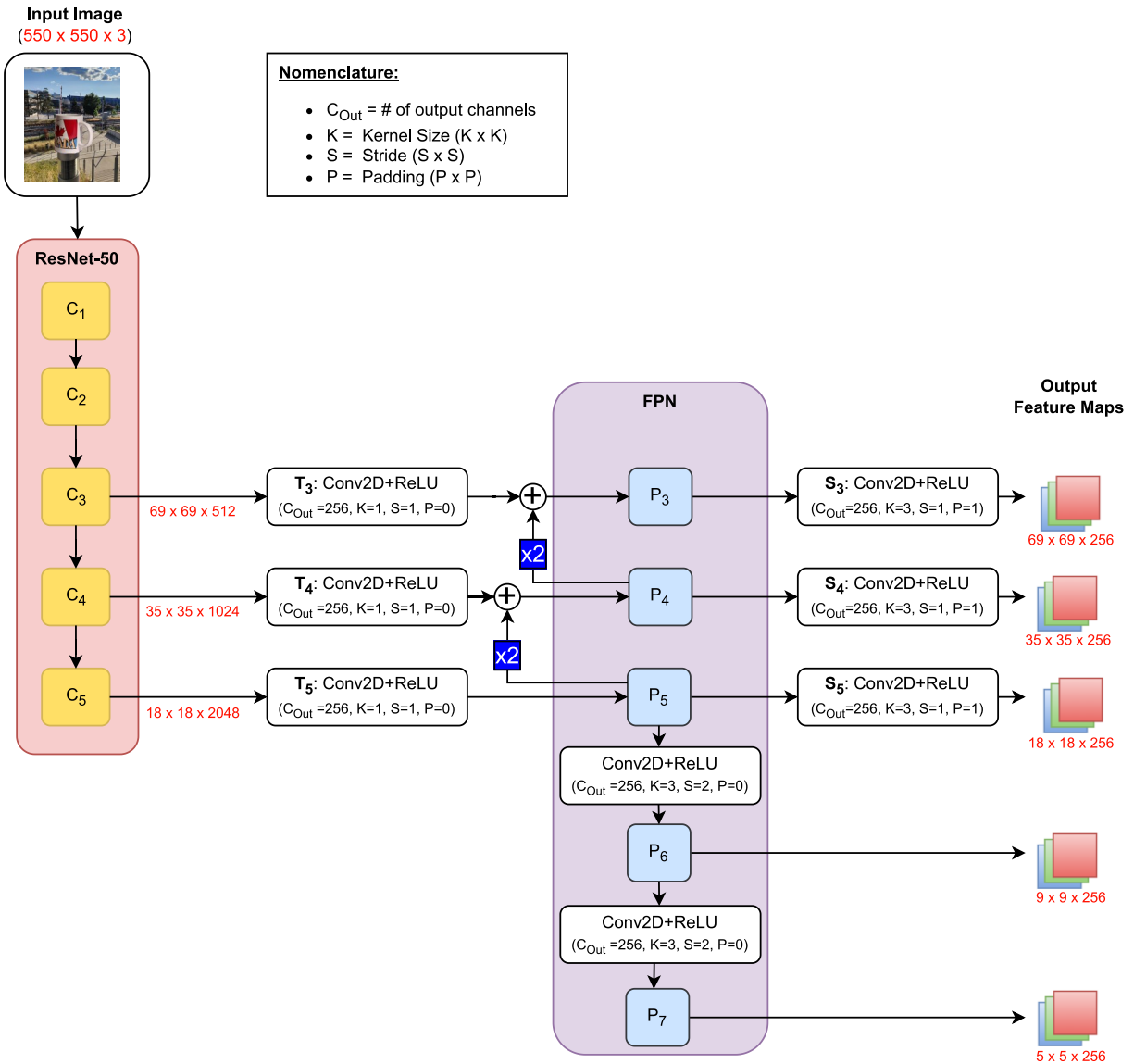
Feature Map Level ( $P_N$ )	Feature Map Dimensions ( $W \times H$ )	Stride ( $x \times y$ )	Anchor Size	Anchor Aspect Ratios
$P_3$	$69 \times 69$	$8 \times 8$	24	1:1, 1:2, 2:1
$P_4$	$35 \times 35$	$16 \times 16$	48	
$P_5$	$18 \times 18$	$32 \times 32$	96	
$P_6$	$9 \times 9$	$64 \times 64$	192	
$P_7$	$5 \times 5$	$128 \times 128$	384	

In YOLACT [82], only  $P_3$  (with the largest feature map) was used to generate mask prototypes for image mask predictions [82]. The optimal number of mask prototypes to balance both speed and accuracy was 32, as stated in [82]. The Protonet has a similar architecture to the mask prediction head in Mask R-CNN [33], except only three convolutional layers are used before the feature map is upsampled with bilinear interpolation to double its size. Two convolutional layers proceed, before the feature map channel dimension is reduced to 32 for the total number of prototype image masks. The generated image mask prototypes do not have an associated object class at this stage, as it only shows the objectness value for each pixel in the resized image. Unlike Mask R-CNN [33], YOLACT [82] predicts the image masks on the entire image, instead of on each object from a given ROI.

The prediction head independently determines the predictions for each level in the FPN. For each feature map from the FPN, two convolutional layers with 256 output channels were applied, before the object classes, bounding boxes, and the mask coefficients were predicted in the corresponding  $H_C$ ,  $H_B$ , and  $H_M$  submodules of  $H_N$ . The mask coefficients are the weights that instruct the model on how the prototype masks should be assembled to produce an image mask. Each weight value can be any real number that is assigned to each mask prototype. Through a linear combination of the weighted mask prototypes, the binary image masks for each corresponding object class are assembled [82]. Before the assembly of the binary image masks, the maximum number of instances is reduced to 200, by using NMS with a threshold of 0.5. The final binary image masks are cropped from the image, and then filtered with binary thresholding to remove the background noise.



**Figure 3.22:** Architecture of the YOLACT model adapted from [82]. All image dimensions (in red) are listed in the following order: number of proposals or instances (if applicable), width, height, and number of channels. The blue  $\times 2$  blocks indicate an upsampling layer, doubling the previous feature map size.



**Figure 3.23:** Modified architecture of the ResNet-50 FPN Network for YOLACT, adapted from [82]. All image dimensions (in red) are listed in the following order: width, height, and number of channels. The blue  $\times 2$  blocks indicate an upsampling layer, doubling the previous feature map size.

### 3.4.3 YOLOv5s-Seg

The YOLOv5s-Seg [83] model (Fig. 3.24) is an extended version of the YOLOv5 object detection model with the mask coefficient and mask assembly prediction head adapted from YOLACT [82]. All initial input images in YOLOv5s-Seg were resized to  $640 \times 640$  pixels. The backbone of YOLOv5s-Seg used the CSP-Darknet53 architecture instead of the ResNet-50 FPN network, as seen in Mask R-CNN and YOLACT. Each of the  $C3$  blocks functions like the ResNet-50 convolutional blocks ( $C_N$ ) for feature extraction in the backbone network. In addition, the  $C3$  blocks assist the transition to a higher resolution feature map during upsampling, which functions similarly to an FPN structure. Unlike the ResNet-50 FPN based models, YOLOv5s-Seg replaced all ReLU activation functions with Sigmoid Linear Units (SiLU).

As seen in Fig. 3.25, the  $C3$  blocks followed the architecture of a Cross Stage Partial Network (CSPNet) [85]. The base layer before each  $C3$  block splitted the incoming inputs into two parallel networks: one subnetwork with a single convolutional layer with batch normalization, and another that followed the architecture of the convolutional blocks in the DarkNet53 model. The first subnetwork was composed of a convolutional layer with batch normalization, and a bottleneck module. The two parallel subnetworks were then concatenated along the channel dimension.

The YOLOv5s-Seg model had two similar modules that function like an FPN in ResNet-50 FPN. The first module was the Spatial Pyramid Pooling Fusion (SPPF) block (Fig. 3.25) that immediately followed the backbone network. Unlike the FPN in ResNet-50 FPN, the SPPF block only received one input from the last layer of the backbone network. The input feature mapped to the SPPF block passed through three max pooling layers, which gradually reduced the resolution of a feature map without reducing the size of the feature map. The feature maps produced by the max pooling layers maintained the size of the feature maps at  $20 \times 20$  pixels; however, the details of the new feature maps gradually got coarser to represent the lower-level features from a given RGB image. The outputs from each of the max pooling layers were then concatenated along the channel dimension. The concatenated feature maps went through a final convolutional layer with batch normalization to return the output channel dimension to 1024. The second FPN-like structure occurred at the neck of the YOLOv5s-Seg model that followed after the SPPF module. The feature map from the SPPF module was upsampled two times, doubling the size of the feature map each time. Similar to ResNet-50 FPN, a skip connection from the backbone network concatenated the outputs of the  $C3$  blocks to the similarly sized upsampling layers (with the same number of channels and



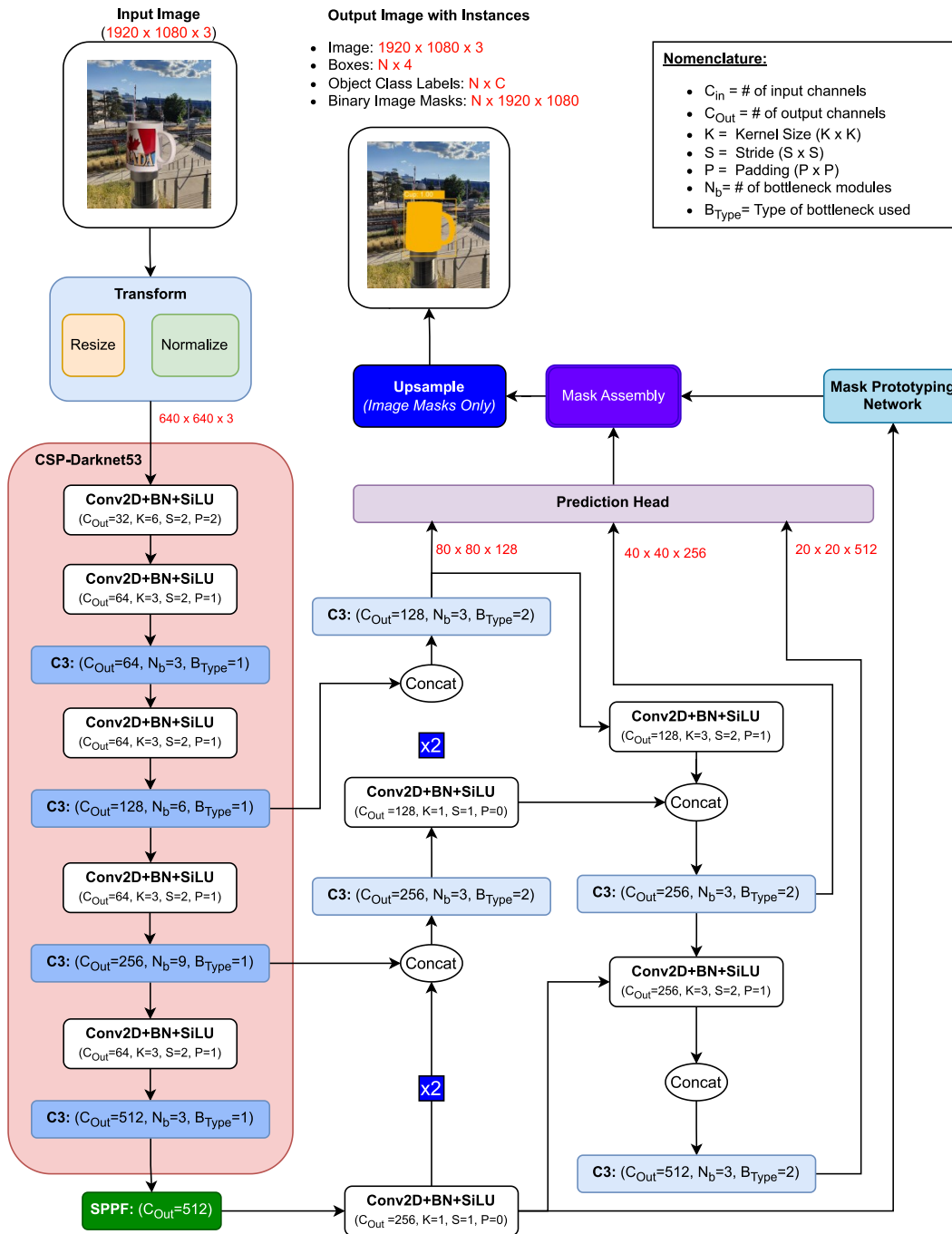
feature map size) in the neck. The layers connected in the skip connections were concatenated by summation. In YOLOv5s-Seg [83], only three unique scales of the feature maps ( $20 \times 20$ ,  $40 \times 40$ ,  $80 \times 80$ ) were used to make bounding box and object class predictions.

During bounding box predictions (the prediction head module in Fig. 3.26), the anchors were generated by using the same method proposed in Mask R-CNN [33] and YOLACT [82]. However, three aspect ratios and three unique anchor sizes (instead of one anchor size) were chosen for each feature map, as shown in Table 3.7. Similar to YOLACT [82], the bounding boxes were predicted independently on each of the three feature maps at the three unique scales, except that the convolutional layers were replaced with *C3* blocks; for each feature map, there was a corresponding prediction head for the object class, bounding boxes, and mask coefficients. The final prediction layer was a single convolutional layer without any activation functions.

**Table 3.7:** YOLOv5s-Seg [83] anchor specifications for each feature map produced before the prediction head. The stride values were rounded up to the nearest pixel, and the anchor sizes were automatically adjusted based on the collective dimensions of the input images in the MOMA dataset.

<b>Feature Map Level</b>	<b>Feature Map Dimensions (<math>W \times H</math>)</b>	<b>Stride (<math>x \times y</math>)</b>	<b>Anchor Sizes (<math>W \times H</math>)</b>
1	$80 \times 80$	$8 \times 8$	$10 \times 13$ , $16 \times 30$ , $33 \times 23$
2	$40 \times 40$	$16 \times 16$	$30 \times 61$ , $62 \times 45$ , $59 \times 119$
3	$20 \times 20$	$32 \times 32$	$116 \times 90$ , $156 \times 198$ , $373 \times 326$

During image mask prediction (mask prototyping network in Fig. 3.26), 32 initial mask prototypes were generated by using the feature map with the largest size ( $80 \times 80$ ) with a similar architecture as ProtoNet in YOLACT [82]. The mask prototyping network was comprised of 5 convolutional layers, where the feature map was upsampled to  $160 \times 160$  in between the second and third convolutional layers. The maximum number of instances detected was reduced to 300 with NMS, before the binary image masks were assembled by using the weights of the mask coefficients.



**Figure 3.24:** Architecture of the YOLOv5s-Seg model adapted from [83]. All image dimensions (in red) are listed in the following order: width, height, and number of channels. The blue  $\times 2$  blocks indicate an upsampling layer, doubling the previous feature map size, while the *Concat* blocks concatenate the feature maps along the channel dimension.

- Nomenclature:**
- $C_{in}$  = # of input channels
  - $C_{Out}$  = # of output channels
  - $K$  = Kernel Size ( $K \times K$ )
  - $S$  = Stride ( $S \times S$ )
  - $P$  = Padding ( $P \times P$ )
  - $N_b$  = # of bottleneck modules
  - $B_{Type}$  = Type of bottleneck used

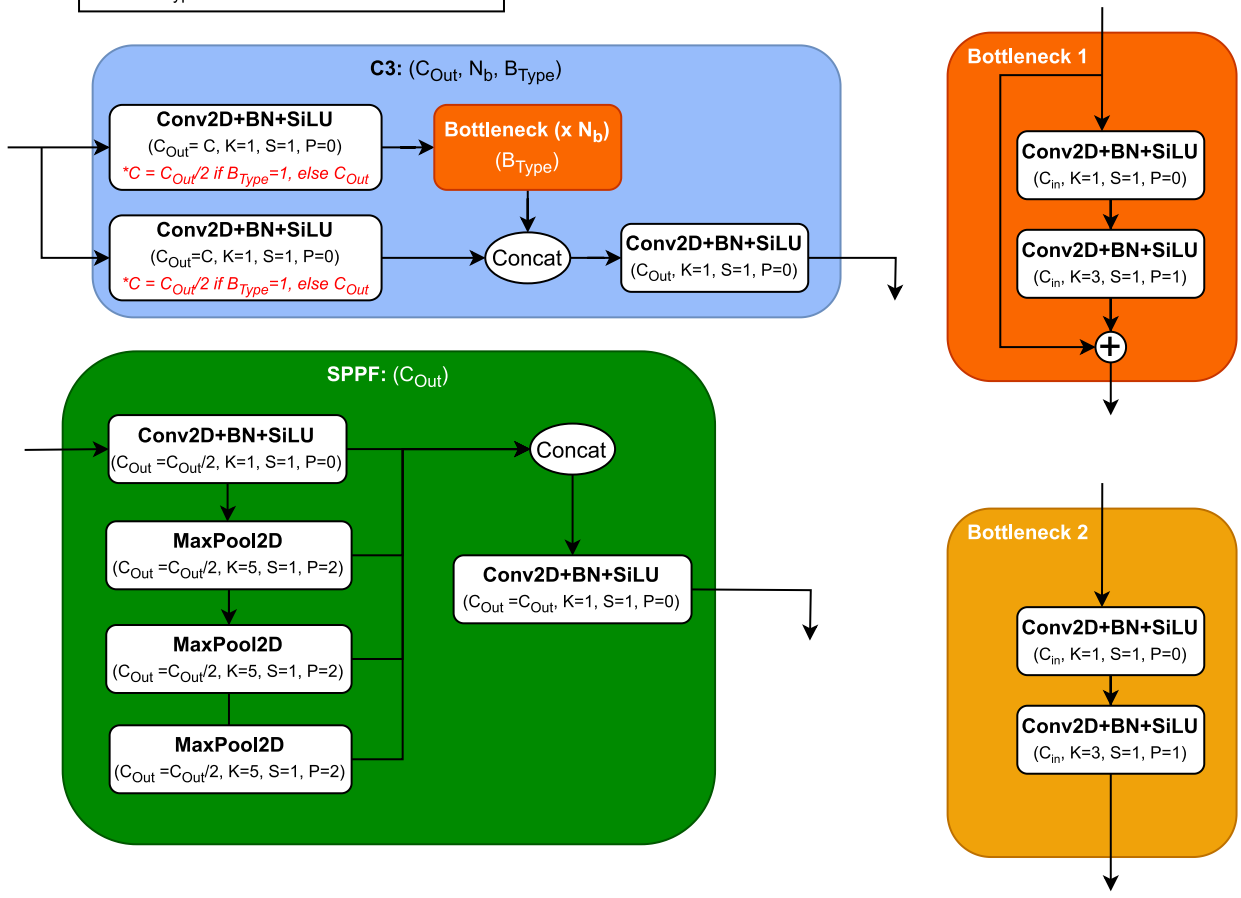
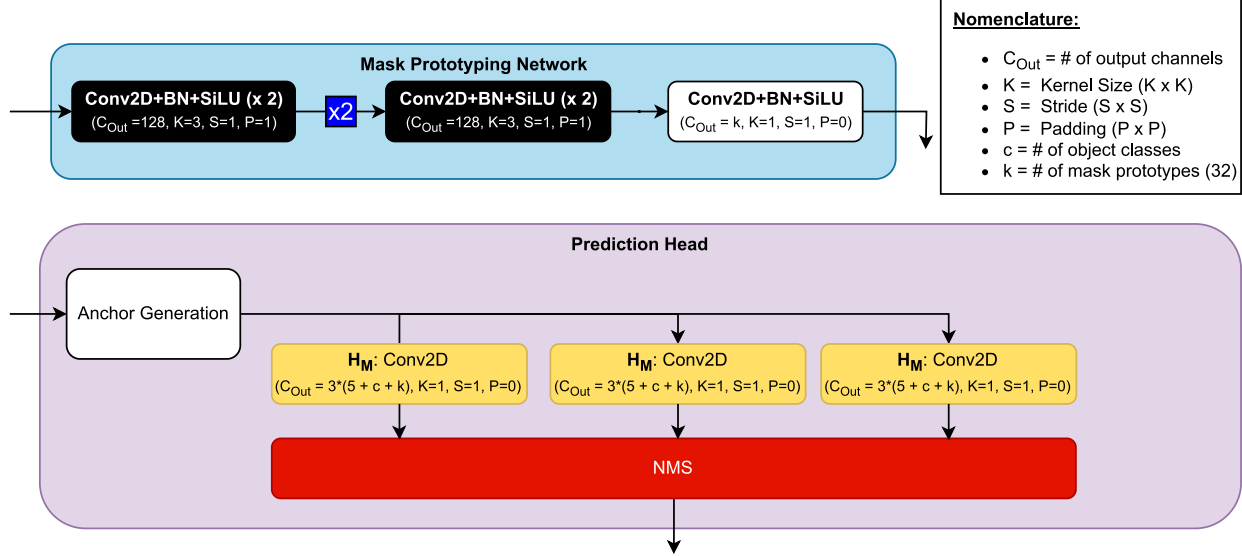


Figure 3.25: Basic building blocks of the YOLOv5s-Seg model, adapted from [83].



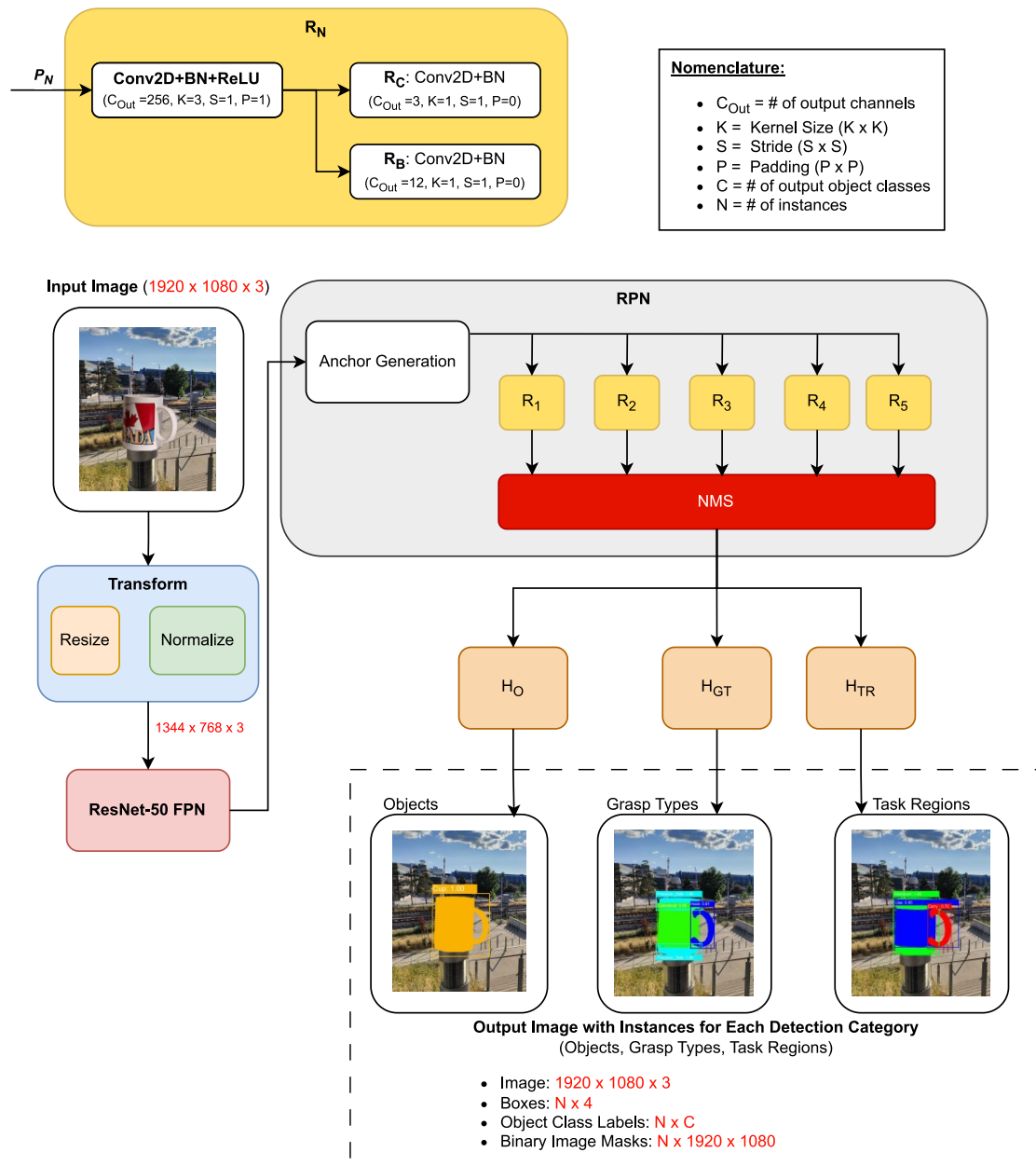
**Figure 3.26:** Network architecture of the mask prototyping network and prediction head of the YOLOv5s-Seg model [83].

### 3.4.4 Multi-Head Mask R-CNN for MAD-Net

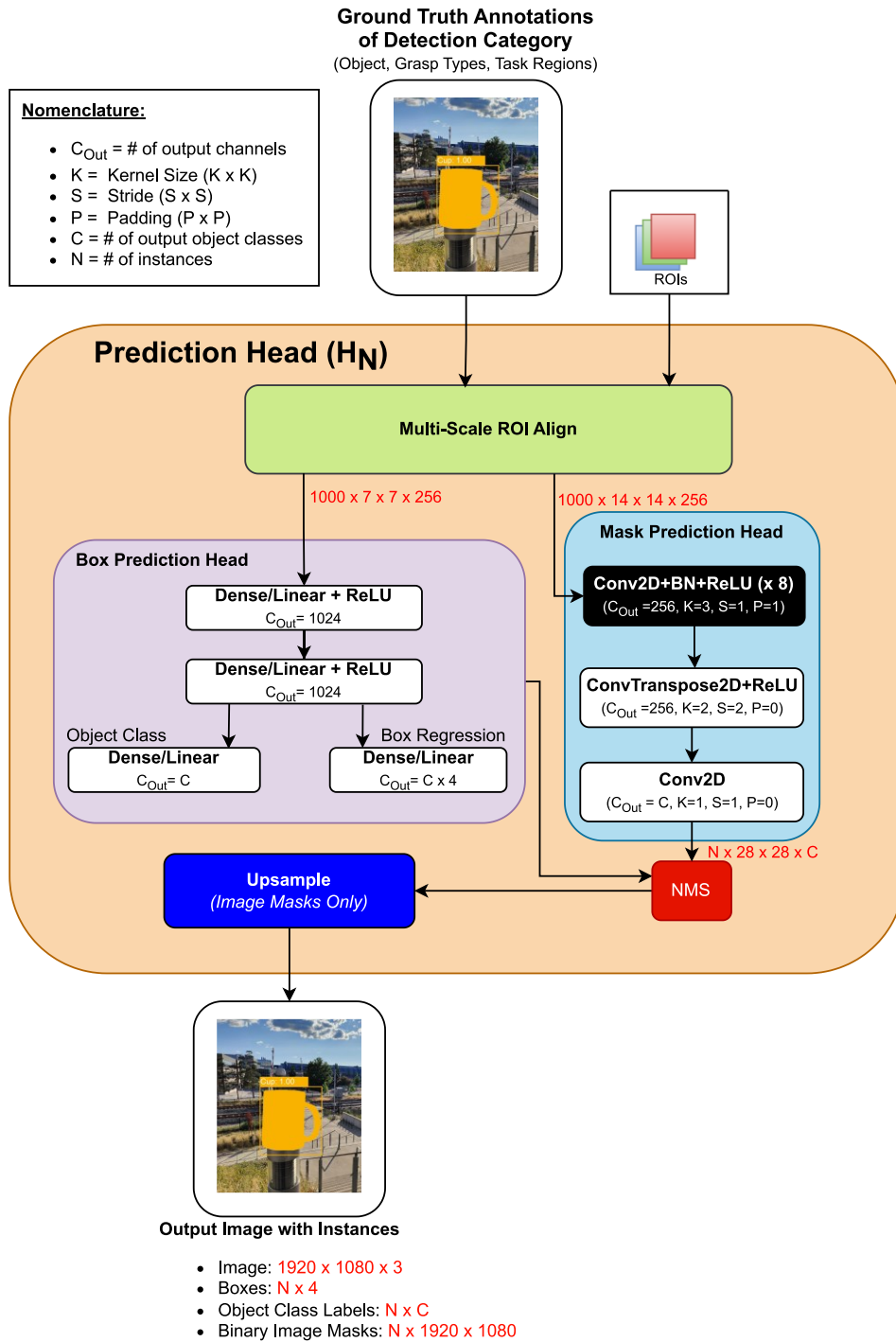
The Multi-head Mask R-CNN model (Fig. 3.27) was a modification of Mask R-CNN with  $N$  prediction heads, where  $N$  is the number of detection categories. Since the Multi-Head Mask R-CNN model was used for the detections of objects, grasp types, and task regions, three prediction heads were used ( $N = 3$ ). The purpose of developing a joint object-and-affordance detection network was to demonstrate the feasibility that the detected object instances can positively impact the accuracy in predicting the grasp affordances, as grasp-type and task affordances. The Mask R-CNN model was expected to perform well in detecting instances of objects since it was already pre-trained on the COCO 2017 dataset [65]. Therefore, by using transfer learning, minimal learning was required to detect the instances of the grasp affordances.

As seen in Fig. 3.27, the ResNet-50 FPN and RPN modules remain unmodified since the object and grasp-affordance classes were not defined at this stage. After the ROIs were reduced from NMS, each of the three prediction heads ( $H_O, H_{GT}, H_{TR}$ ) received a copy of the ROIs. Fig. 3.28 shows the network architecture of the prediction head. As a result, the object and their grasp affordances could be inferred separately if required. Each of the prediction heads predicted the bounding boxes, object or grasp affordance class, and the binary image masks using the same mask and box head in Mask R-CNN. During

training, the loss values across all prediction heads were independently computed, but jointly optimized by summing all losses. In addition, the corresponding annotation categories were also matched with the correct detection category of the prediction head (e.g., grasp-type annotations were compared with the grasp-type predictions).



**Figure 3.27:** Architecture of the MAD-Net model. All image dimensions (in red) are listed in the following order: number of proposals or instances (if applicable), width, height, and number of channels.



**Figure 3.28:** Prediction Head for the detection of object classes, bounding boxes, and binary image masks in MAD-Net.

### 3.5 Experimental Procedure for Training and Evaluating on the MOMA Synthetic Dataset

The deep-learning models described in Sec. 3.4 were all trained using the Nvidia GeForce RTX 3060 graphics card, which has 12 GB of available memory. All models were initially trained for 10 epochs. If the training and validation losses did not converge by the end of training, the training was resumed in intervals of 5 epochs. During training, all models were optimized with an SGD optimizer that has a learning rate of  $1 \times 10^{-3}$  and momentum of 0.9.

Before the RGB images and the ground truth annotations were fed into the models for training, the images were additionally augmented with a function that randomly rotates the image 90 degrees clockwise, counterclockwise, or remaining the same. The probability of the images rotating by 90 degrees was set to 0.5 (0.25 each for both clockwise and counterclockwise directions). The rotation of the images simulated the artificially changing egocentric viewpoint on the objects as discussed in Sec. 3.2.1. In addition, the extra rotation transformation diversified the small collection of unique background images used in the MOMA synthetic dataset. For Mask R-CNN, YOLACT, and Multi-Head Mask R-CNN (MAD-Net), the annotations were in the COCO JSON format, while YOLOv5s-Seg had the annotations in YOLO TXT format.

During training, Mask R-CNN and MAD-Net used a batch size of 2 images, while YOLACT and YOLOv5s-Seg used a batch size of 8 images. For Mask R-CNN, YOLACT, and YOLOv5s-Seg, the training, validation, and evaluation process was repeated three times for each detection category (objects, grasp types, and task regions). On the other hand, MAD-Net was trained and evaluated in a single session with all detection categories. During evaluation, all test images were inferenced first before the evaluation metrics described in Sec. 3.5.2 were used to evaluate model performance.

For all models, the data used to train, validate, and test were sourced from the MOMA dataset. During training, the Train-S set was used, and during validation, the Val-S1 set was used. Only the loss functions specified in Sec. 3.5.1 were used to evaluate model performance during training and validation. For evaluation, all validation and test sets (Val-S1, Val-S2, Test-S, Test-I) were evaluated separately with the evaluation metric specified in Sec. 3.5.2.

### 3.5.1 Training Loss Criterion for Object and Grasp-Affordance Detection with Instance Segmentation

During training and validation, the computed losses were obtained for every batch of  $N$  images that got passed into the inputs of the instance segmentation models. The total sum of all losses from all image batches was the resulting loss value for one epoch.

The objective function for Mask R-CNN is given below in Eq. 3.9 [68],

$$L_T = L_{class} + L_{box} + L_{mask} + L_{rpn} \quad (3.9)$$

where  $L_{class}$  is the object class loss,  $L_{box}$  is the box loss,  $L_{mask}$  is the mask loss, and  $L_{rpn}$  is the total loss from the RPN during training and validation.  $L_{class}$  is the cross-entropy loss computed with Eq. 3.3 from Sec. 3.1.2, by comparing the predicted and ground truth object classes. The predicted classes in Mask R-CNN were a 1D vector of logits, containing the probabilities for each object class.  $L_{box}$  and  $L_{mask}$  were computed in the box and mask heads, respectively. Using Eq. 3.10,  $L_{box}$  was calculated with the smooth L1 loss function [68] since the prediction of bounding boxes is a regression problem. The bounding boxes in Mask R-CNN was a  $B \times 4$ , 2D vector with  $B$  boxes, where each box was in the form:

$[x_{1_b}, y_{1_b}, x_{2_b}, y_{2_b}]$ .  $b$  was used to represent the selected 1D box coordinates, since  $n$  was already used in Eq. 3.10 [68] to describe the  $n_{th}$  image in a batch of  $N$  images.

$$\mathbf{L}_{box_n} = \mathbf{L}_{SL_{1n}} = \begin{cases} 0.5(\mathbf{x}_n - \mathbf{y}_n)^2 / \beta, & |\mathbf{x}_n - \mathbf{y}_n| < \beta \\ |\mathbf{x}_n - \mathbf{y}_n| - 0.5\beta, & else \end{cases} \quad (3.10)$$

$$L_{box} = L_{SL_1} = \sum_{n=1}^N L_{SL_{1n}} = \sum_{n=1}^N \sum_{i,j} (\mathbf{L}_{SL_{1n}})_{ij} \quad (3.11)$$

In Eq. 3.10,  $\mathbf{x}_n$  and  $\mathbf{y}_n$  are the predicted and target bounding box locations for image  $n$ , respectively. Eq. 3.10 computes the Smooth L1 loss vector,  $\mathbf{L}_{box_n}$ , for all bounding boxes predicted in image  $n$ ; and the error difference tolerance value,  $\beta$ , was set to 1/9 for Mask R-CNN. The error difference between  $\mathbf{x}_n$  and  $\mathbf{y}_n$  was penalized more if it was less than  $\beta$  to ensure that all errors in the predictions (minor or major) were weighted similarly. In Eq. 3.11 [68],  $\mathbf{L}_{box_n}$  was reduced to a single loss value by summing all of its elements, before the total box loss,  $L_{box}$ , was calculated. The sum of all box losses for each image in a batch of  $N$  images was the total box loss. To calculate the mask loss,  $L_{mask}$ , the binary cross-entropy loss function [68] was used. The calculation of the binary cross-entropy losses was similar to Eq. 3.3 from



Sec. 3.1.2. However, the sigmoid function was used in-place of the softmax function since the object class label on the mask could be either 1 or 0 (objectness). The objects existed on an image if a mask logit had a value greater than 0.5. The binary cross-entropy losses were calculated using Eqs. 3.12, 3.13, and 3.14 [68].

$$\sigma(\mathbf{X}_n) = \frac{1}{1 + \exp(\mathbf{X}_n)} \quad (3.12)$$

$$\mathbf{L}_{mask_n} = \mathbf{L}_{BCE_n} = -[\mathbf{Y}_n \ln \sigma(\mathbf{X}_n) + (1 - \mathbf{Y}_n) \ln(1 - \sigma(\mathbf{X}_n))] \quad (3.13)$$

$$L_{mask} = L_{BCE} = \sum_{n=1}^N \frac{L_{BCE_n}}{\text{count}(\mathbf{Y}_n)} = \sum_{n=1}^N \frac{\sum_{i,j} (\mathbf{L}_{BCE_n})_{ij}}{\text{count}(\mathbf{Y}_n)} \quad (3.14)$$

Eq. 3.12 is the sigmoid function that receives the predicted binary image mask,  $\mathbf{X}_n$  for image  $n$ . The resulting 2D matrix was evaluated with the target image mask,  $\mathbf{Y}_n$  in Eq. 3.13. Eq. 3.13 calculates  $\mathbf{L}_{BCE_n}$ , the binary cross-entropy loss matrix for image  $n$ . In Eq. 3.14,  $\mathbf{L}_{BCE_n}$  is first reduced by taking the average of each element in the loss matrix to find the single loss value,  $L_{BCE_n}$ , for image  $n$ . All  $L_{BCE_n}$  loss values for all images in batch  $N$  are then summed together to obtain the total mask loss for batch  $N$ . The objective function of Mask R-CNN also calculates the loss values from the RPN,  $L_{RPN}$ .  $L_{RPN}$  is determined by summing the box and mask losses (Eqs. 3.11 and 3.14) by the end of the RPN and the five feature maps produced by the ResNet-50 FPN backbone, respectively.

For MAD-Net, the same objective function for Mask R-CNN (Eq. 3.9) was used to calculate the training and validation losses. Since MAD-Net simultaneously detects the instances for the objects, grasp types, and task regions, the losses were independently calculated for each detection category. Therefore, the objective function for MAD-Net was:

$$L_T = L_O + L_{GT} + L_{TS} \quad (3.15)$$

where  $L_O$  is the total loss for the detection of the objects (cups, bottles, etc.),  $L_{GT}$  is the total loss for the detection of grasp types, and  $L_{TS}$  is the total loss for the detection of the task regions.

The objective function for YOLACT [82], adapted from SSD [55], was a linear combination of the smooth L1 losses and the cross-entropy losses, as shown in Eq. 3.16 [55, 82]. The cross-entropy losses

represent the losses during classification, while the smooth L1 losses represent the losses during regression.

$$L_T = \frac{1}{M} [L_{CE_T} + L_{SL1_T}], \quad (3.16)$$

where  $M$  is the number of matching boxes with the target boxes,  $L_{CE_T}$  is the total cross-entropy (and binary cross-entropy) losses and  $L_{SL1_T}$  is the total smooth L1 losses. The total cross-entropy and smooth L1 losses for YOLACT [82] are given in Eqs. 3.17 and 3.18.

$$L_{CE_T} = \sum_n^N \alpha_1 L_{class_n} + \alpha_2 L_{seg_n} + \alpha_3 L_{mask_n} \quad (3.17)$$

$$L_{SL1_T} = \alpha \sum_n^N L_{box_n} \quad (3.18)$$

In Eq. 3.17, for image  $n$ ,  $L_{class_n}$  is the class loss during the classification of object classes,  $L_{seg_n}$  is the segmentation loss during the mask prototype generation at  $P_3$ , and  $L_{mask_n}$  is the mask loss from the resulting binary image masks during mask assembly.  $L_{class_n}$  is the same as calculating the cross-entropy losses over all possible object classes (Eq. 3.3), while  $L_{mask_n}$  and  $L_{seg_n}$  are calculated as a binary cross-entropy loss (Eq. 3.14).  $L_{CE_T}$  in Eq. 3.17 therefore finds the total cross-entropy and binary cross-entropy losses in a batch of  $N$  images. The class and segmentation losses were weighted equally, with  $\alpha_1 = \alpha_2 = 1$ , while the errors during mask assembly was weighted higher with  $\alpha_3 = 6.125$ . In Eq. 3.18,  $L_{box_n}$  represents the box loss for a single image  $n$ . Since the bounding boxes were predicted through regression, only the box loss was calculated with the smooth L1 loss function (Eq. 3.11). The box loss was also similarly weighted with  $\alpha = 1.5$ . Overall, the mask loss and box losses had a higher penalty cost, since the localization of the boxes and the accuracy in the segmented binary masks were more significant than simply predicting if the instance was accurately classified.

For YOLOV5s-Seg [83], the objective loss function was similar to YOLACT; however, the loss from predicting the bounding boxes was treated as both a regression and classification problem. The following equation was used to calculate the total box loss for a batch of  $N$  images:

$$L_{box} = L_{CIOU} + L_{obj}, \quad (3.19)$$

$L_{obj}$  is the objectness loss that describes if the predicted bounding box is accurately placed where an object exists, while  $L_{CIOU}$  is the Complete Intersection Over Union (CIOU) losses adapted from [86].  $L_{CIOU}$  is similar to finding the smooth L1 losses like in Mask R-CNN and YOLACT; however,  $L_{CIOU}$  also penalizes the model if the aspect ratios between the predicted and target boxes do not match. Therefore,  $L_{CIOU}$  enforces the predicted bounding boxes to have a similar aspect ratio, centre coordinates, and IOU as the matched ground truth bounding boxes. In YOLOV5s-Seg [83], both class and mask losses are computed as binary cross-entropy losses (Eq. 3.14). The final losses are the sum of the box, class, and mask losses, except that the total loss is divided by the number of images in a batch. Therefore, the final objective function for YOLOV5s-Seg [83] is:

$$L_T = \frac{1}{N}(L_{box} + L_{mask} + L_{class}), \quad (3.20)$$

where  $N$  is the total number of images in a batch,  $L_{box}$  is the total box loss,  $L_{mask}$  is the mask loss, and  $L_{class}$  is the class loss.

### 3.5.2 Evaluation Metrics

Before the models were evaluated by an evaluation metric, the images from all validation and test sets in the MOMA dataset were inferenced. All predictions with a confidence score in the object class predictions (objectness probability) that had a value of at least 0.5 were kept. When a prediction was discarded, the corresponding boxes and binary image masks were also discarded. The predictions of the binary image masks were further filtered, so that all mask logits with a value greater than 0.5 were considered to contain an object.

During the evaluation of all models, the mean average precision (mAP) metric was used since model performance was dependent on a model's ability to localize the detected instances. The mAP scores were determined for each detection category (objects, grasp types, and task regions) and for each of the validation and test sets from the MOMA dataset (4 in total). The mAP scores were also calculated twice – once for the bounding box predictions, and once for the binary mask predictions. For each image, the precision and recall scores were first determined using the following equations:

$$P = \frac{TP}{TP + FP} \quad (3.21)$$

$$R = \frac{TP}{TP + FN} \quad (3.22)$$

where  $P$  and  $R$  are the precision and recall scores, respectively; and  $TP$ ,  $FP$ , and  $FN$  are the number of true positives, false positives, and false negatives, respectively. A true positive occurred when the mIOU between the predicted instances and the matching target instances (as a bounding box or as an image mask) was at least 0.5. In addition, the object class for the matched instances had to be the same, otherwise, the prediction was a false positive. All unmatched instances between the predictions and the targets were the false negatives. During the calculation of the precision and recall curves, the scores were separated by object class; thus, each object class received a precision-recall score.

Once the precision and recall scores were calculated for each image and each object class, the precision-recall curve was plotted. The mAP score was then obtained by finding the area-under-the-curve (AUC) of the precision-recall curve. The average of all mAP scores across all object classes was then computed.

## Chapter 4

# Results and Discussion of the Instance Segmentation Models Trained on the MOMA Dataset

For the experimental results presented in this chapter, the Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], and MAD-Net models were trained, evaluated, and inferenced (by creating prediction images from an input image) on the MOMA dataset. The primary objective was to determine the feasibility of training the detection of objects with the grasp affordances, in applications to prosthetic grasping. Therefore, the joint detection model, MAD-Net, was evaluated against the other models (Mask R-CNN [33], YOLACT [82], and YOLOV5s-Seg [83]) for the quality of the inference images and mAP scores on the MOMA dataset. Overall, each model was trained to detect the instances of objects, grasp types, and task regions.

Sec. 4.1 presents the resulting training and validation loss curves of the models that were initially trained for 10 epochs. The training session was completed only if both the training and validation losses had stably converged (no random loss explosion). In Secs. 4.2.1 and 4.2.2, the evaluated mAP scores are compared for each detection category (objects, grasp types, and task regions) across all the validation and test sets of the MOMA dataset. Sec. 4.2.1 compares the mAP scores during bounding box detection, and Sec. 4.2.2 compares the mAP scores during binary image segmentation of the detected instances. Sec. 4.2.3 and Sec. 4.2.4 then give the resulting prediction images during inferencing for the Test-S and Test-I datasets, respectively. Sec. 4.2.5 provides the overall summary of the model performance during training and evaluation, including the overall training time, GPU usage, and the average validation and test mAP score. Finally, Sec. 4.3 concludes Chapter 4 to discuss the factors that may have contributed to how each model performed, and the feasibility of the detection of object and its grasp affordances with instance segmentation.

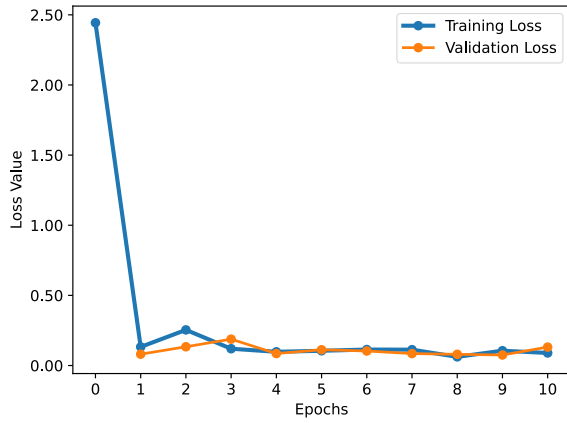
### 4.1 Experimental Evaluation of the Instance Segmentation Models during Training

The Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], and MAD-Net models were trained and validated for each epoch during the training sessions. The mAP scores were not evaluated during training and validation since the loss values were enough to determine whether the model had completed training.

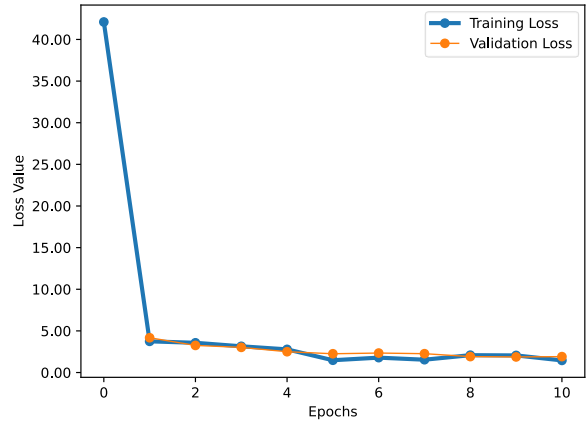
#### 4.1.1 Training and Validation Loss Curves

The training and validation loss curves for the detection of object, grasp-type, and task-region instances are given in Figs. 4.1, 4.2, and 4.3, respectively. When training the models for the detection of object instances, the Mask R-CNN [33], YOLACT [82], and MAD-Net models only required 10 epochs to complete training, whereas YOLOv5s-Seg [83] required 20 epochs. In most cases, the validation losses matched the training losses for all models across all detection categories (Fig. 4.1). Note that the initial 0<sup>th</sup> epoch in all training-validation loss plots are the loss values during the weight initialization of the models. When training the models for the detection of grasp-type instances, the Mask R-CNN model was trained for 15 epochs, since there was an unexpected spike in the validation losses by the end of 10 epochs (Fig. 4.2a). The YOLACT and MAD-Net models were trained for 10 epochs each, while the YOLOv5s-Seg model similarly completed training after 20 epochs. During the training sessions for the detection of the task-region categories (Fig. 4.3), the Mask R-CNN, YOLACT, MAD-Net models all required 10 epochs to complete training, and YOLOv5s-Seg required 20 epochs.

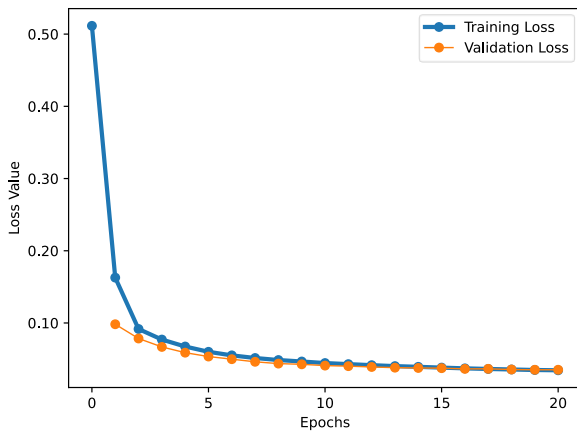
For the YOLOv5s-Seg model, the learning rate had an initial value of 0.0 before it reached the original learning rate of  $1 \times 10^{-3}$  by the second epoch. The resulting loss graphs were smoother than the other models in Figs. 4.1c to 4.3c. YOLACT similarly had the warmup learning rate, starting from  $1 \times 10^{-4}$  and ending at  $1 \times 10^{-3}$ ; however, the resulting training and validation losses still converged similarly to Mask R-CNN and MAD-Net, except that the validation losses remained stable without the sudden loss explosion.



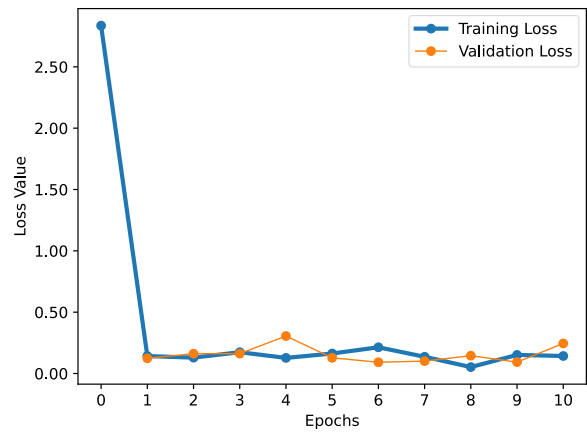
(a)



(b)

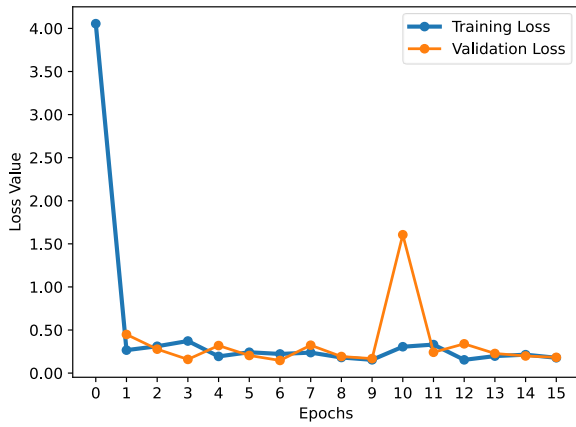


(c)

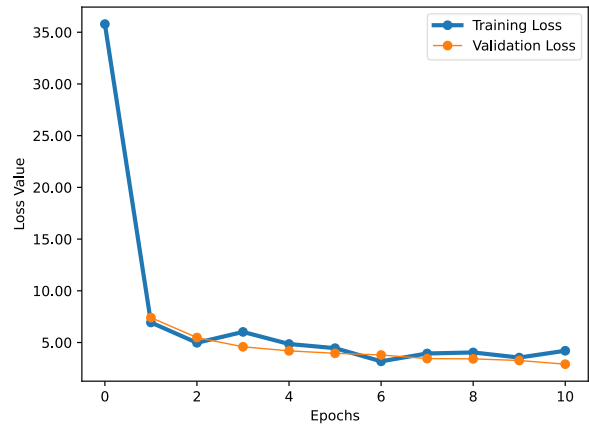


(d)

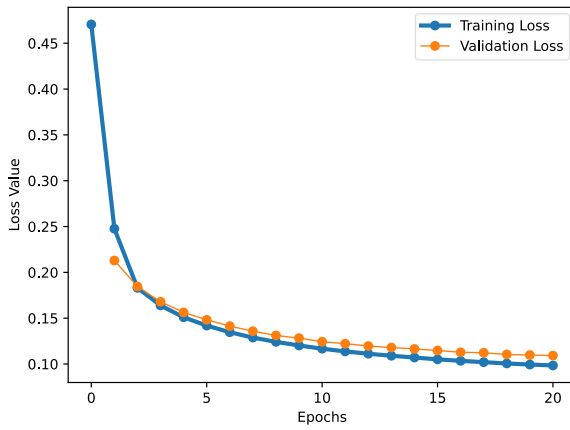
**Figure 4.1:** Training and validation plots for detection of instances of objects (cups, bottles, etc.): (a) Mask R-CNN, (b) YOLACT, (c) YOLOv5s-Seg, and (d) MAD-Net.



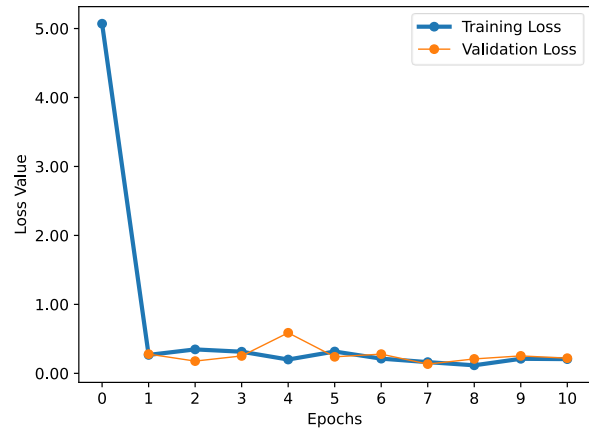
(a)



(b)



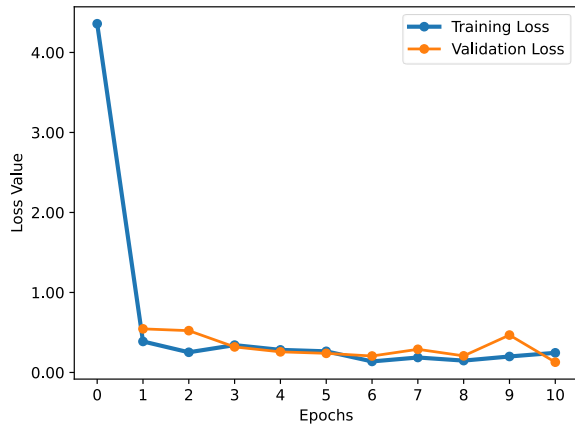
(c)



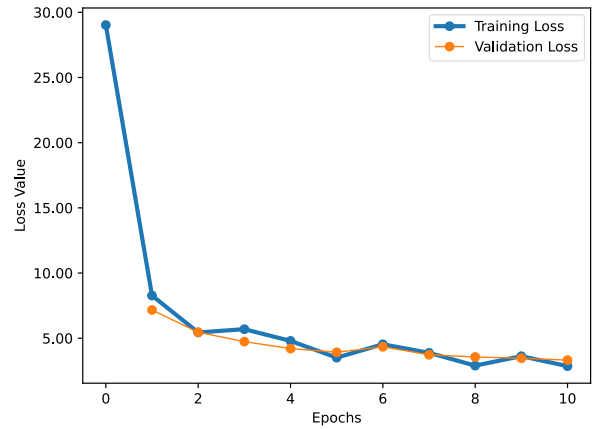
(d)

**Figure 4.2:** Training and validation plots for detection of instances of grasp types: (a) Mask R-CNN, (b) YOLACT, (c) YOLOv5s-Seg, and (d) MAD-Net.

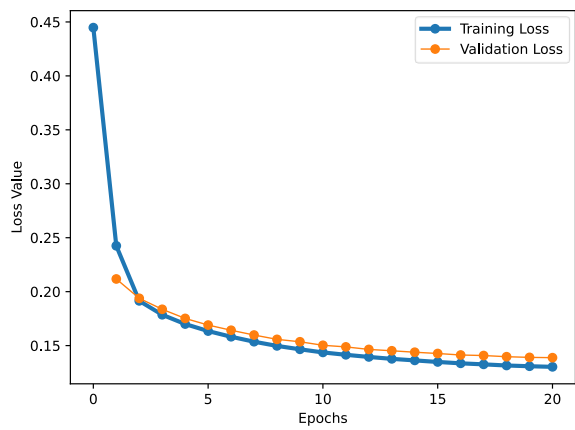




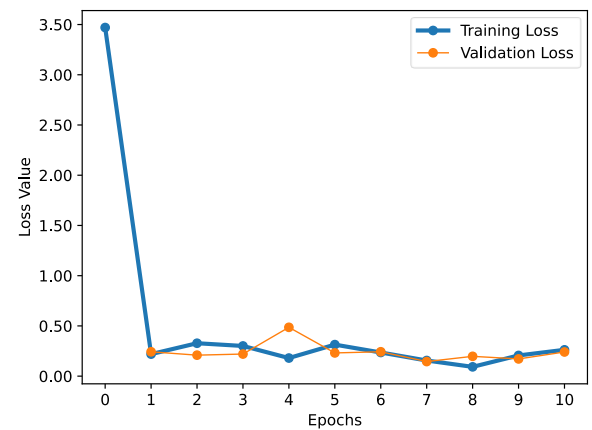
(a)



(b)



(c)



(d)

**Figure 4.3:** Training and validation plots for detection of instances of task regions: (a) Mask R-CNN, (b) YOLACT, (c) YOLOv5s-Seg, and (d) MAD-Net.

## 4.2 Experimental Evaluation of the Instance Segmentation Models during Testing

Secs. 4.2.1 and 4.2.2 presents the mAP evaluation scores for the Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], and MAD-Net models. Sec. 4.2.1 gives the mAP scores during bounding box detection, while Sec. 4.2.2 provides the mAP scores for binary image mask segmentation. The models tested were evaluated on their ability to simultaneously detect the instances of objects, grasp types, and task regions on new and familiar objects. The minimum mIOU threshold for a true positive detection as an image mask or bounding box was set to 0.5.

### 4.2.1 mAP Evaluation Results for Bounding Box Detection

The final bounding box detection, mAP evaluation results for the tested models (Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], and MAD-Net) are given in Tables 4.1 to 4.3. Overall, the MAD-Net model had the best average mAP score across all datasets and detection categories, except for the detection of task-region instances where YOLOv5s-Seg had a better average mAP score. All the tested models had a consistent mAP score in all datasets. However, the Mask R-CNN and MAD-Net models did underperform in the detection of the *carry* task in the task-region detection category (Table 4.3). As later seen in Sec. 4.2.4, the inference results show that the Mask R-CNN and MAD-Net models could not simultaneously detect two entirely overlapping instances. Therefore, only one of the instances was kept and labelled with an object category (*use* class in the case of the detection of task-region instances). The mAP scores generally decreased as the models were evaluated on the datasets with more unfamiliar objects. The validation mAP scores were significantly higher than the test mAP scores, especially when the models were evaluated on the Test-I dataset (Tables 4.1 to 4.3). The Test-I dataset was generated to contain completely new objects with unconventional object parts. For example, a wine glass was introduced into the *cups* category, which had an extra stem to *carry* and *use* the object with a *prismatic-4-finger* grasp type.

**Table 4.1:** mAP scores for object bounding box detection (cups, bottles, etc.). The tested models were evaluated on all validation and test sets of the MOMA dataset.

Model	Dataset	mAP Scores by Object Class at mIOU $\geq 0.5$				
		Bottle	Cup	Hammer	Knife	All
Mask R-CNN	Val-S1	0.997	0.990	0.999	0.991	0.994
	Val-S2	0.989	0.980	0.990	0.982	0.985
	Test-S	0.986	0.962	0.978	0.894	0.955
	Test-I	0.914	0.791	0.831	0.803	0.834
	<b>Average mAP</b>	<b>0.972</b>	<b>0.931</b>	<b>0.949</b>	<b>0.918</b>	<b>0.942</b>
	MAD-Net	Val-S1	1.000	0.998	0.999	0.997
	Val-S2	0.998	0.996	0.997	0.993	0.996
	Test-S	0.995	0.978	0.991	0.894	0.965
	Test-I	0.943	0.827	0.859	0.749	0.845
	<b>Average mAP</b>	<b>0.984</b>	<b>0.950</b>	<b>0.962</b>	<b>0.908</b>	<b>0.951</b>
	YOLOACT	Val-S1	0.946	0.979	0.830	0.729
	Val-S2	0.944	0.968	0.882	0.712	0.877
	Test-S	0.865	0.934	0.813	0.609	0.805
	Test-I	0.865	0.934	0.813	0.609	0.805
	<b>Average mAP</b>	<b>0.905</b>	<b>0.953</b>	<b>0.834</b>	<b>0.665</b>	<b>0.839</b>
	YOLOV5s-Seg	Val-S1	0.995	0.989	0.978	0.951
	Val-S2	0.989	0.986	0.973	0.954	0.975
	Test-S	0.976	0.971	0.921	0.882	0.938
	Test-I	0.868	0.848	0.760	0.870	0.837
	<b>Average mAP</b>	<b>0.957</b>	<b>0.949</b>	<b>0.908</b>	<b>0.914</b>	<b>0.932</b>

**Table 4.2:** mAP scores for grasp-type bounding box detection. The tested models were evaluated on all validation and test sets of the MOMA dataset.

Model	Dataset	mAP Scores by Grasp-Type Class at mIOU $\geq 0.5$							
		Adducted Thumb	Cylindrical	Hook	Lateral Tripod	Light Tool	Precision Disk	Prismatic 4 Finger	All
Mask R-CNN	Val-S1	0.931	0.853	0.860	0.788	0.858	0.841	0.905	0.862
	Val-S2	0.937	0.830	0.888	0.784	0.799	0.795	0.915	0.850
	Test-S	0.729	0.741	0.798	0.812	0.901	0.729	0.673	0.769
	Test-I	0.443	0.381	0.287	0.436	0.477	0.483	0.373	0.411
	<b>Average mAP</b>	<b>0.760</b>	<b>0.701</b>	<b>0.708</b>	<b>0.705</b>	<b>0.759</b>	<b>0.712</b>	<b>0.717</b>	<b>0.723</b>
	MAD-Net	Val-S1	0.996	0.989	0.994	0.997	0.997	0.991	0.991
	Val-S2	0.968	0.919	0.960	0.747	0.952	0.946	0.969	0.923
	Test-S	0.809	0.932	0.920	0.962	0.979	0.939	0.789	0.904
	Test-I	0.500	0.483	0.320	0.483	0.561	0.744	0.439	0.504
	<b>Average mAP</b>	<b>0.818</b>	<b>0.831</b>	<b>0.799</b>	<b>0.797</b>	<b>0.872</b>	<b>0.905</b>	<b>0.797</b>	<b>0.831</b>
	YOLOACT	Val-S1	0.843	0.943	0.854	0.913	0.788	0.915	0.771
	Val-S2	0.769	0.891	0.849	0.758	0.783	0.837	0.746	0.805
	Test-S	0.662	0.840	0.848	0.722	0.748	0.781	0.548	0.736
	Test-I	0.662	0.840	0.848	0.722	0.748	0.781	0.548	0.736
	<b>Average mAP</b>	<b>0.734</b>	<b>0.879</b>	<b>0.850</b>	<b>0.779</b>	<b>0.767</b>	<b>0.829</b>	<b>0.653</b>	<b>0.784</b>
	YOLOV5s-Seg	Val-S1	0.800	0.947	0.920	0.862	0.931	0.954	0.801
	Val-S2	0.780	0.914	0.889	0.855	0.893	0.908	0.793	0.862
	Test-S	0.624	0.923	0.889	0.925	0.855	0.913	0.619	0.821
	Test-I	0.371	0.548	0.376	0.500	0.585	0.738	0.519	0.520
	<b>Average mAP</b>	<b>0.644</b>	<b>0.833</b>	<b>0.769</b>	<b>0.786</b>	<b>0.816</b>	<b>0.878</b>	<b>0.683</b>	<b>0.773</b>

**Table 4.3:** mAP scores for task-region bounding box detection. The tested models were evaluated on all validation and test sets of the MOMA dataset.

Model	Dataset	mAP Scores by Task-Region Class at mIOU $\geq 0.5$			
		Carry	Handover	Use	All
Mask R-CNN	Val-S1	0.617	0.992	0.996	0.869
	Val-S2	0.626	0.970	0.970	0.855
	Test-S	0.581	0.936	0.946	0.829
	Test-I	0.509	0.826	0.655	0.664
	<b>Average mAP</b>	<b>0.583</b>	<b>0.931</b>	<b>0.892</b>	<b>0.804</b>
MAD-Net	Val-S1	0.617	0.994	0.997	0.869
	Val-S2	0.621	0.973	0.966	0.854
	Test-S	0.565	0.931	0.941	0.812
	Test-I	0.509	0.799	0.651	0.653
	<b>Average mAP</b>	<b>0.578</b>	<b>0.924</b>	<b>0.889</b>	<b>0.797</b>
YOLACT	Val-S1	0.812	0.889	0.877	0.859
	Val-S2	0.778	0.870	0.848	0.832
	Test-S	0.711	0.834	0.818	0.788
	Test-I	0.711	0.834	0.818	0.788
	<b>Average mAP</b>	<b>0.753</b>	<b>0.856</b>	<b>0.840</b>	<b>0.817</b>
YOLOV5s-Seg	Val-S1	0.921	0.958	0.954	0.944
	Val-S2	0.897	0.933	0.946	0.925
	Test-S	0.864	0.904	0.919	0.895
	Test-I	0.777	0.819	0.666	0.754
	<b>Average mAP</b>	<b>0.865</b>	<b>0.904</b>	<b>0.871</b>	<b>0.880</b>

#### 4.2.2 mAP Evaluation Results for Binary Image Mask Segmentation

The final binary image segmentation, mAP evaluation results for the tested models (Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], and MAD-Net) are given in Tables 4.4 to 4.6. Similarly to Sec. 4.2.1, the MAD-Net model had the best average mAP score across all datasets and detection categories; the MAD-Net model was able to outperform YOLOv5s-Seg model in all cases except for the segmentation of the *carry* image masks in Table 4.6. Overall, all the tested models also had a consistent mAP score in all datasets, and a similar decreasing trend in performance when the models were evaluated on the test sets. However, the YOLOv5s-Seg model had a significant mAP score decrease in the segmentation of the task-region image masks for the *use* task. The Mask R-CNN and MAD-Net models

had a similar performance drop in the detection of the *carry* task (Table 4.6). YOLACT generally had a consistent mAP score in all datasets for all detection categories (objects, grasp types, task regions).

**Table 4.4:** mAP scores for object, binary image mask segmentation (cups, bottles, etc.). The tested models were evaluated on all validation and test sets of the MOMA dataset.

Model	Dataset	mAP Scores by Object Class at mIOU $\geq 0.5$				
		Bottle	Cup	Hammer	Knife	All
Mask R-CNN	Val-S1	0.997	0.990	0.999	0.991	0.994
	Val-S2	0.945	0.956	0.900	0.928	0.932
	Test-S	0.986	0.962	0.981	0.894	0.956
	Test-I	0.911	0.802	0.831	0.798	0.836
	<b>Average mAP</b>	<b>0.960</b>	<b>0.927</b>	<b>0.928</b>	<b>0.903</b>	<b>0.929</b>
MAD-Net	Val-S1	1.000	0.998	0.998	0.997	0.998
	Val-S2	0.998	0.996	0.997	0.993	0.996
	Test-S	0.998	0.982	0.972	0.926	0.970
	Test-I	0.942	0.845	0.853	0.744	0.846
	<b>Average mAP</b>	<b>0.984</b>	<b>0.955</b>	<b>0.955</b>	<b>0.915</b>	<b>0.953</b>
YOLACT	Val-S1	0.947	0.979	0.830	0.738	0.873
	Val-S2	0.944	0.968	0.882	0.713	0.877
	Test-S	0.866	0.934	0.813	0.619	0.808
	Test-I	0.866	0.934	0.813	0.619	0.808
	<b>Average mAP</b>	<b>0.906</b>	<b>0.953</b>	<b>0.834</b>	<b>0.672</b>	<b>0.841</b>
YOLOV5s-Seg	Val-S1	0.995	0.989	0.981	0.953	0.979
	Val-S2	0.989	0.986	0.973	0.950	0.975
	Test-S	0.976	0.971	0.925	0.882	0.938
	Test-I	0.868	0.858	0.764	0.866	0.839
	<b>Average mAP</b>	<b>0.957</b>	<b>0.951</b>	<b>0.911</b>	<b>0.913</b>	<b>0.933</b>

**Table 4.5:** mAP scores for grasp-type, binary image mask segmentation. The tested models were evaluated on all validation and test sets of the MOMA dataset.

Model	Dataset	mAP Scores by Grasp-Type Class at mIOU $\geq 0.5$							
		Adducted Thumb	Cylindrical	Hook	Lateral Tripod	Light Tool	Precision Disk	Prismatic 4 Finger	All
Mask R-CNN	Val-S1	0.993	0.880	0.958	0.974	0.970	0.921	0.959	0.951
	Val-S2	0.936	0.792	0.886	0.781	0.800	0.738	0.905	0.834
	Test-S	0.730	0.672	0.796	0.758	0.901	0.667	0.681	0.744
	Test-I	0.457	0.341	0.283	0.449	0.518	0.435	0.376	0.408
	<b>Average mAP</b>	<b>0.779</b>	<b>0.671</b>	<b>0.731</b>	<b>0.740</b>	<b>0.797</b>	<b>0.690</b>	<b>0.730</b>	<b>0.734</b>
	MAD-Net	Val-S1	0.994	0.988	0.993	0.997	0.997	0.981	0.980
	Val-S2	0.968	0.902	0.957	0.710	0.953	0.905	0.958	0.907
	Test-S	0.850	0.921	0.933	0.945	0.984	0.811	0.824	0.895
	Test-I	0.510	0.437	0.320	0.467	0.652	0.648	0.437	0.496
	<b>Average mAP</b>	<b>0.831</b>	<b>0.812</b>	<b>0.801</b>	<b>0.780</b>	<b>0.897</b>	<b>0.836</b>	<b>0.800</b>	<b>0.822</b>
	YOLOACT	Val-S1	0.843	0.917	0.745	0.845	0.801	0.814	0.746
	Val-S2	0.770	0.857	0.741	0.660	0.794	0.731	0.724	0.754
	Test-S	0.675	0.827	0.801	0.647	0.769	0.701	0.568	0.713
	Test-I	0.675	0.827	0.801	0.647	0.769	0.702	0.568	0.713
	<b>Average mAP</b>	<b>0.740</b>	<b>0.857</b>	<b>0.772</b>	<b>0.700</b>	<b>0.783</b>	<b>0.737</b>	<b>0.652</b>	<b>0.749</b>
	YOLOV5s-Seg	Val-S1	0.800	0.912	0.889	0.810	0.928	0.906	0.790
	Val-S2	0.778	0.880	0.865	0.810	0.893	0.845	0.781	0.836
	Test-S	0.625	0.901	0.866	0.788	0.851	0.835	0.615	0.783
	Test-I	0.385	0.516	0.370	0.513	0.636	0.685	0.517	0.517
	<b>Average mAP</b>	<b>0.647</b>	<b>0.802</b>	<b>0.748</b>	<b>0.730</b>	<b>0.827</b>	<b>0.818</b>	<b>0.676</b>	<b>0.750</b>

**Table 4.6:** mAP scores for task-region, binary image mask segmentation. The tested models were evaluated on all validation and test sets of the MOMA dataset.

Model	Dataset	mAP Scores by Task-Region Class at mIOU $\geq 0.5$			
		Carry	Handover	Use	All
Mask R-CNN	Val-S1	0.617	0.991	0.996	0.868
	Val-S2	0.558	0.898	0.842	0.766
	Test-S	0.577	0.930	0.946	0.818
	Test-I	0.506	0.812	0.704	0.674
	<b>Average mAP</b>	<b>0.564</b>	<b>0.908</b>	<b>0.872</b>	<b>0.781</b>
MAD-Net	Val-S1	0.617	0.994	0.996	0.869
	Val-S2	0.619	0.969	0.954	0.848
	Test-S	0.577	0.926	0.949	0.817
	Test-I	0.513	0.782	0.705	0.667
	<b>Average mAP</b>	<b>0.582</b>	<b>0.918</b>	<b>0.901</b>	<b>0.800</b>
YOLACT	Val-S1	0.757	0.863	0.861	0.827
	Val-S2	0.722	0.855	0.820	0.799
	Test-S	0.703	0.822	0.812	0.779
	Test-I	0.703	0.822	0.812	0.779
	<b>Average mAP</b>	<b>0.721</b>	<b>0.840</b>	<b>0.826</b>	<b>0.796</b>
YOLOV5s-Seg	Val-S1	0.695	0.671	0.086	0.484
	Val-S2	0.668	0.632	0.071	0.457
	Test-S	0.623	0.661	0.081	0.455
	Test-I	0.603	0.530	0.071	0.401
	<b>Average mAP</b>	<b>0.647</b>	<b>0.624</b>	<b>0.077</b>	<b>0.449</b>

### 4.2.3 Inferencing Results for the Test-S Set in the MOMA Dataset

The Test-S set of the MOMA dataset contains unique objects and background images that differ from the training set (Train-S) that the models (Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], MAD-Net) were trained on. However, since the Train-S and Test-S sets of the MOMA dataset used objects from the standard object set, the objects in the Test-S set had similar features as the objects in the Train-S set. As a result, all models, except YOLOV5s-Seg, precisely localized the instances with the corresponding bounding boxes and binary image masks across all detection categories. The inferencing results for the detection of object and grasp-type instances, are given in Figs. 4.4 and 4.5, respectively. For the detection of task-regions, the inferencing results are displayed on separate images by task class, since the task



regions on an object excessively overlap one another. The inferring results for the detection of task-region instances are shown in Figs. 4.6, 4.7, and 4.8.



**Figure 4.4:** Instance segmentation predictions for the object (bottles, cups, etc.) detection category: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.



**Figure 4.5:** Instance segmentation predictions for the grasp-type detection category: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.





**Figure 4.6:** Instance segmentation predictions for the task-region detection category, for the *carry* class: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.



**Figure 4.7:** Instance segmentation predictions for the task-region detection category, for the *handover* class: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.





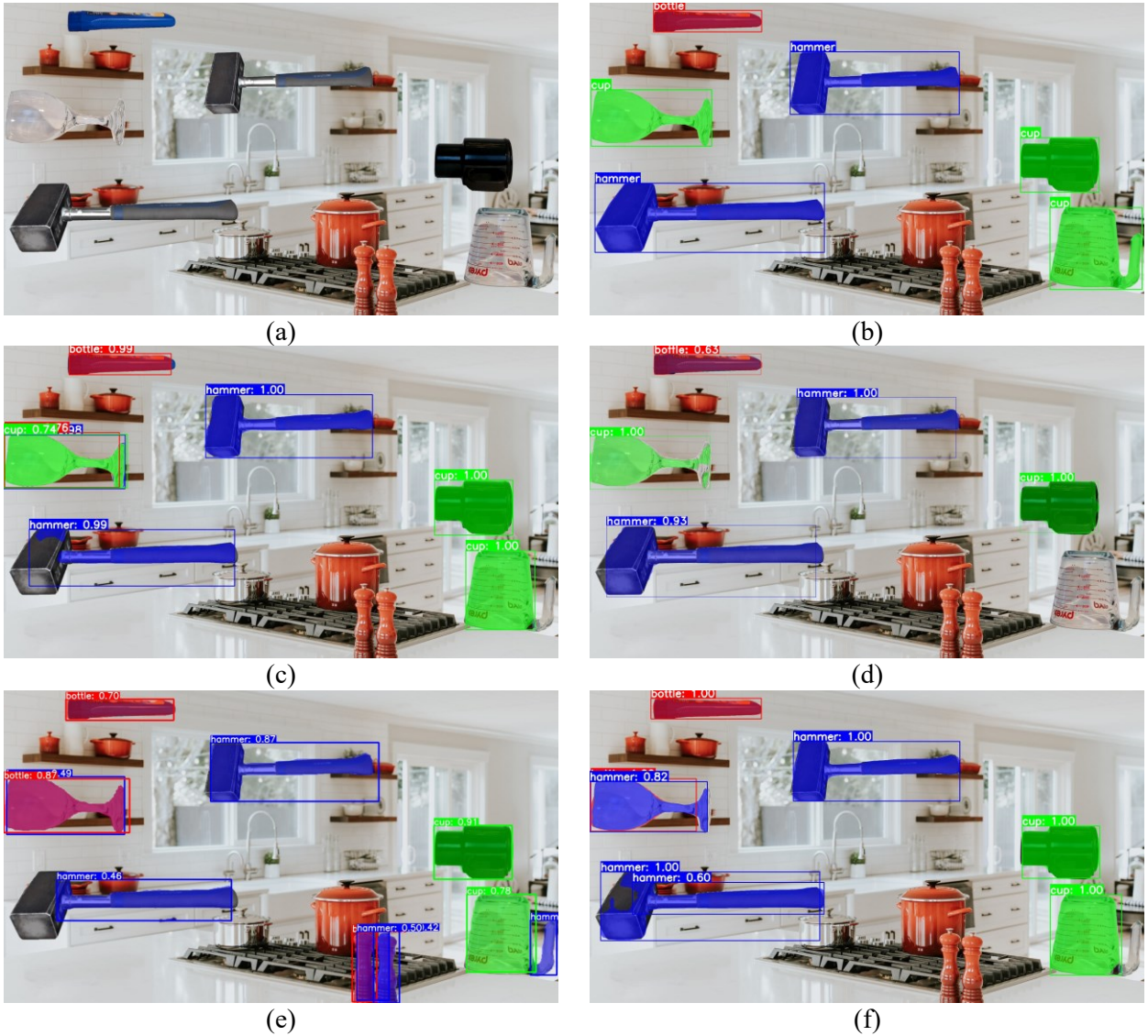
**Figure 4.8:** Instance segmentation predictions for the task-region detection category, for the *use* class: (a) original input image, and (b) ground truth or target. (c) to (f) are the predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.

#### 4.2.4 Inferencing Results for the Test-I Set in the MOMA Dataset

The Test-I set of the MOMA dataset contained scenes of irregular objects on new backgrounds that were unencountered by the trained models (Mask R-CNN [33], YOLACT [82], YOLOV5s-Seg [83], MAD-Net).

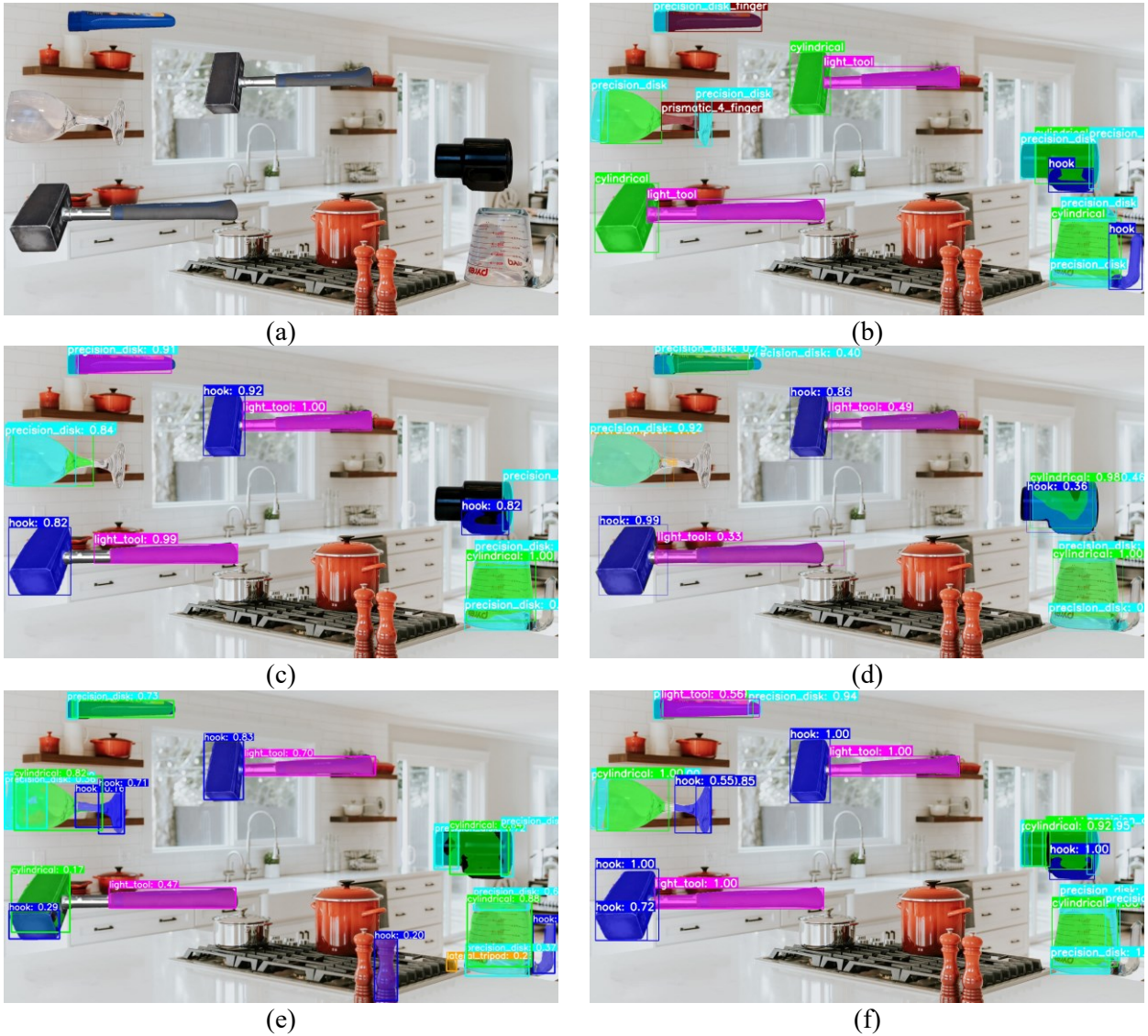
During the detection of object instances on the Test-I Set of the MOMA dataset, all models had consistent results in the inference images in Fig. 4.9. Overall, the MAD-Net model (Fig. 4.9f) had the best image mask quality on the objects, although it occasionally misclassified the object instances. For example, the wine glass in Fig. 4.9f was supposed to have an object class label, *cup*, as indicated by the ground truth annotations in Fig. 4.9b. For YOLOv5s-Seg, however, the model had a higher tendency to generalize to objects in the background (Fig. 4.9e).

The image mask quality of Mask R-CNN and MAD-Net precisely matched the possible locations for grasp-type and task affordances on each object (Figs. 4.10 and 4.13). Since MAD-Net simultaneously trained the detection of objects and their grasp affordances (grasp types and task regions), the resulting locations of the binary image masks on the object parts always aligned with the detected object image mask. However, since the objects in the scene were unconventional compared to the objects found in the training set, Mask R-CNN and MAD-Net both suffered from the misclassifications of the identified instances, which will ultimately decrease the mAP scores. For YOLACT and YOLOv5s-Seg, the model was generally capable of precisely classifying and localizing the instances within the bounding boxes. However, the overall mask quality in detecting the grasp affordances (Figs. 4.10d to 4.13d) were lacking compared to MAD-Net and Mask R-CNN. Mask R-CNN and MAD-Net were also unable to identify the task-regions of two or more instances that were exactly overlapping on top of each other (Figs. 4.10 and 4.13). For example, the hammer should have both the *carry* and *use* task on its handle as seen in Fig. 4.11b and Fig. 4.13b; however, only the *use* region was correctly identified by Mask R-CNN in Fig. 4.13c, and MAD-Net in Fig. 4.13f.



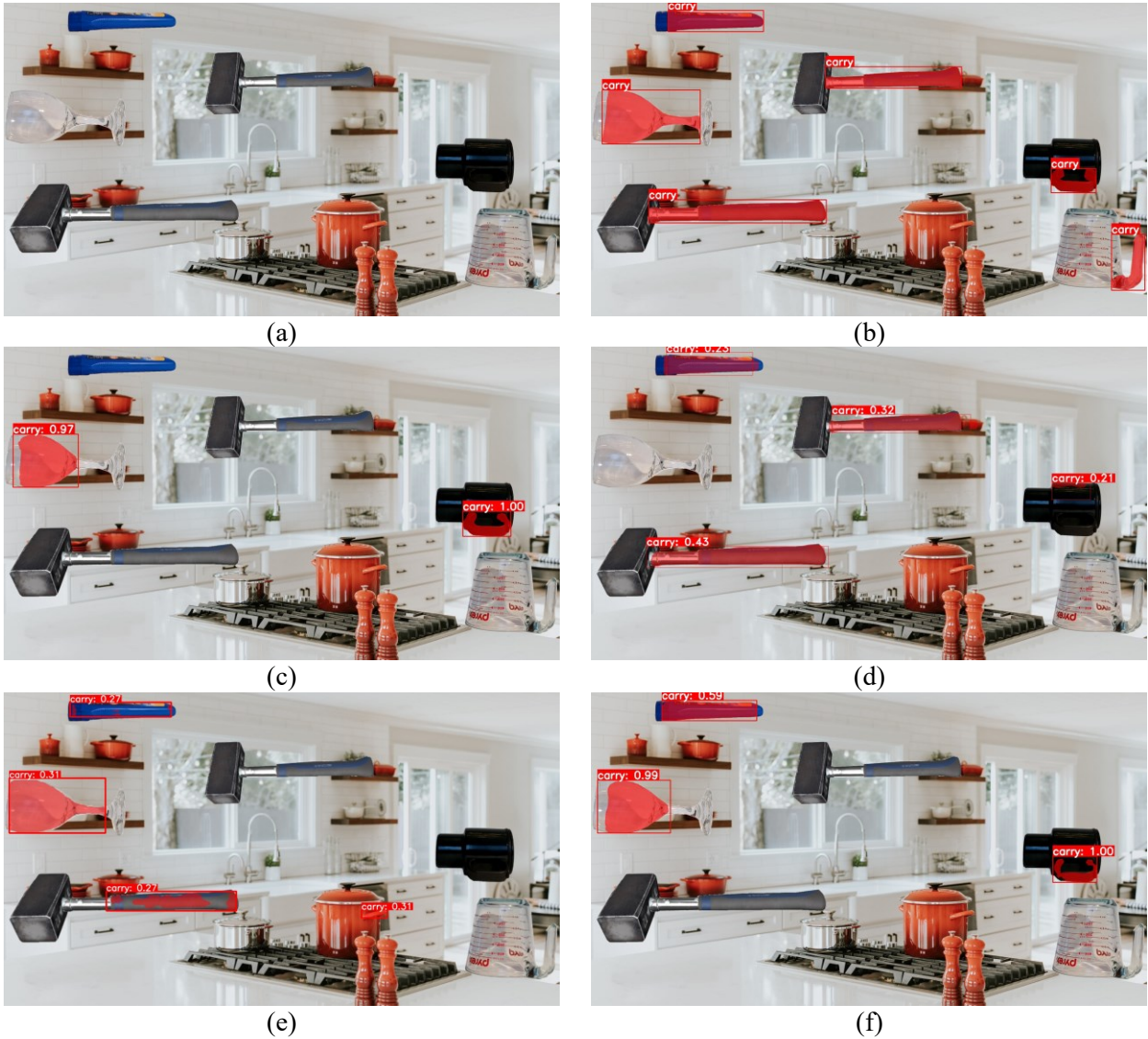
**Figure 4.9:** Instance segmentation predictions for the object (bottles, cups, etc.) detection category: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.



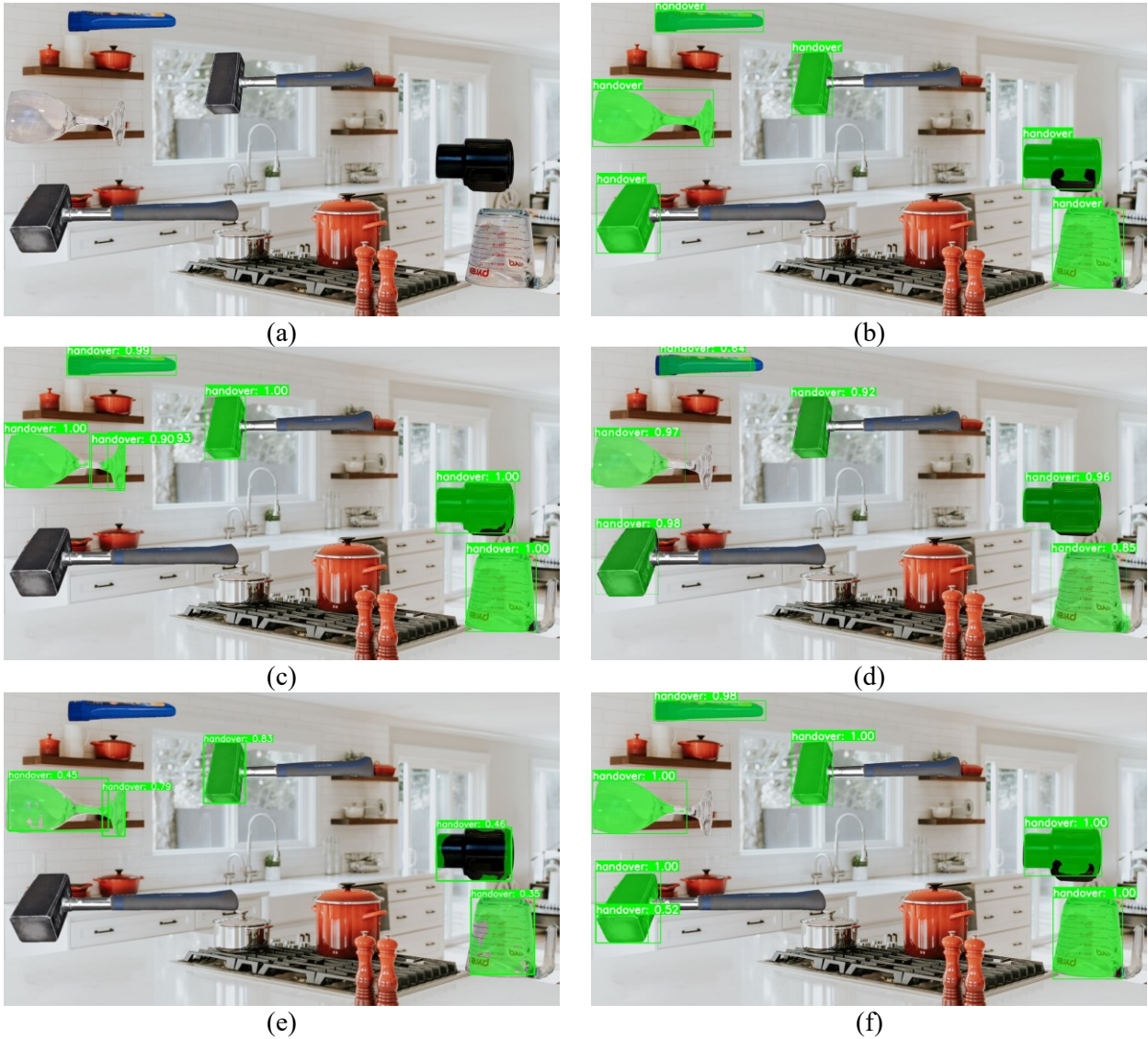


**Figure 4.10:** Instance segmentation predictions for the grasp-type detection category: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.

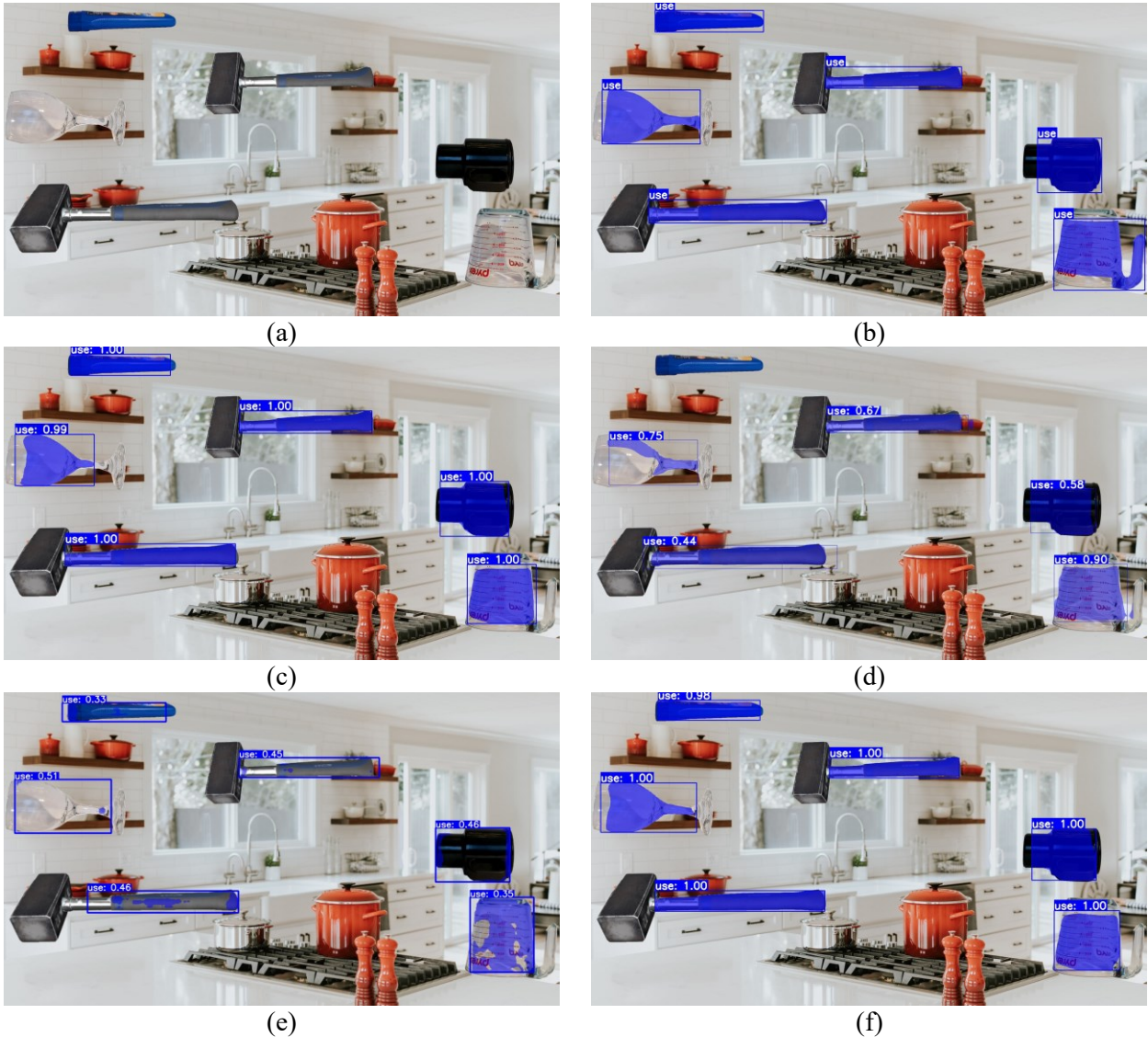




**Figure 4.11:** Instance segmentation predictions for the task-region detection category, for the *carry* class: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.



**Figure 4.12:** Instance segmentation predictions for the task-region detection category, for the *handover* class: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.



**Figure 4.13:** Instance segmentation predictions for the task-region detection category, for the *use* class: (a) original input image, and (b) ground truth or target. (c) to (f) are predictions from Mask R-CNN, YOLACT, YOLOv5s-Seg, and MAD-Net, respectively.

#### 4.2.5 Summary of the Training, Validation, and the Evaluation Results

A summary of the model performances during training, validation, and evaluation is given in Tables 4.7, 4.8, and 4.9. MAD-Net was trained collectively to detect the instances of objects, grasp types, and task regions. Therefore, the elapsed training time in Tables 4.7, 4.8, and 4.9 was the average time that MAD-Net took to train each detection category; the actual training time for MAD-Net was 12:20:47 (hh:mm:ss),



which was surprisingly slightly slower than just training the Mask R-CNN [33] model to detect grasp-type instances. In addition, the MAD-Net model only added a small overhead to train than Mask R-CNN when comparing their overall GPU memory usage.

The YOLOv5s-Seg [83] model trained the fastest and required the least GPU memory to run, while YOLACT [82] took the longest to train. For the grasp-type detection category, however, Mask R-CNN took longer to complete training only because Mask R-CNN needed an extra five epochs to converge the losses. While YOLOv5s-Seg completed training the fastest, it was also underperforming compared to the other models in terms of the mAP score; however, the YOLOv5s-Seg did have the best bounding box predictions during validation and evaluation. In general, the YOLACT model was consistent in the performance, and at times the mAP score for YOLACT outperformed MAD-Net, especially during testing. MAD-Net generally outperformed the other models, especially in the object detection category in Table 4.7. For the detection of grasp-types and task instances, both MAD-Net and Mask R-CNN had a significant decrease in the mAP score because of their poor performance on the Test-I set from the MOMA dataset. Overall, MAD-Net outperformed its Mask R-CNN counterpart in all detection categories.

**Table 4.7:** Results summary for all models trained and evaluated for the detection of object instances (cups, bottles, etc.). The best result values across each category are highlighted in bold. The asterisk (\*) indicates the average time for MAD-Net to complete training for the detection of object instances.

Model	Number of Trained Epochs	Batch Size	Training Time (hh:mm:ss)	GPU Usage, Training (GB)	Average Validation mAP		Average Test mAP	
					Boxes	Masks	Boxes	Masks
Mask R-CNN	10	2	06:28:45	6.00	0.990	0.963	0.895	0.896
YOLACT	10	8	08:26:30	7.30	0.874	0.875	0.805	0.808
YOLOv5s-Seg	20	8	<b>02:38:12</b>	<b>2.71</b>	0.977	0.977	0.888	0.889
MAD-Net	10	2	*04:06:56	8.50	<b>0.997</b>	<b>0.997</b>	<b>0.905</b>	<b>0.908</b>

**Table 4.8:** Results summary for all models trained and evaluated for the detection of grasp-type instances. The best result values across each category are highlighted in bold. The asterisk (\*) indicates the average time for MAD-Net to complete training for the detection of object instances.

Model	Number of Trained Epochs	Batch Size	Training Time (hh:mm:ss)	GPU Usage, Training (GB)	Average Validation mAP		Average Test mAP	
					Boxes	Masks	Boxes	Masks
Mask R-CNN	15	2	11:16:21	7.25	0.856	0.892	0.590	0.576
YOLACT	10	8	07:51:22	7.90	0.833	0.785	<b>0.736</b>	<b>0.713</b>
YOLOv5s-Seg	20	8	<b>02:58:35</b>	<b>3.04</b>	0.875	0.849	0.671	0.650
MAD-Net	10	2	*04:06:56	8.50	<b>0.958</b>	<b>0.949</b>	0.704	0.696

**Table 4.9:** Results summary for all models trained and evaluated for the detection of task-region instances. The best result values across each category are highlighted in bold. The asterisk (\*) indicates the average time for MAD-Net to complete training for the detection of object instances.

Model	Number of Trained Epochs	Batch Size	Training Time (hh:mm:ss)	GPU Usage, Training (GB)	Average Validation mAP		Average Test mAP	
					Boxes	Masks	Boxes	Masks
Mask R-CNN	10	2	06:41:04	6.10	0.862	0.817	0.746	0.746
YOLACT	10	8	08:15:54	8.00	0.846	0.813	0.788	<b>0.779</b>
YOLOv5s-Seg	20	8	<b>03:01:40</b>	<b>3.16</b>	<b>0.935</b>	0.471	<b>0.825</b>	0.428
MAD-Net	10	2	*04:06:56	8.50	0.861	<b>0.858</b>	0.733	0.742

### 4.3 Final Discussion of the Training, Validation, and Testing Results on the MOMA Dataset

Overall, the Mask R-CNN [33], YOLACT [82], YOLOv5s-Seg [83], and MAD-Net models all trained well since the training and validation losses all converged by the end of all epochs. All models were trained using a Nvidia GeForce RTX 3060 graphics card, and all models generally had consistent results in the mAP scores during bounding box detection and binary image segmentation on the MOMA dataset. The resulting mAP scores from Secs. 4.2.1 and 4.2.2 were an accurate indicator to determine a model's performance as shown in the inference images from the test sets in Secs. 4.2.3 and 4.2.4.

The MAD-Net model introduced in this thesis overall had produced better quality image masks, similar to its Mask R-CNN counterpart. The MAD-Net model excelled in all detection categories, in terms of the averaging mAP scores, and displayed a significant improvement over the Mask R-CNN model, in which only a single detection category could be trained and evaluated on. Based on the resulting mAP score improvement from the Mask R-CNN model to the MAD-Net model, the joint detection of the object and its grasp affordances is a feasible solution to implement for a vision system in current prosthetic hands. In addition, the MAD-Net model only required an additional 2.05 GB overhead, on average, to simultaneously train on each detection category (objects, grasp types, task regions). In terms of the overall training time for MAD-Net, the total training time to train all three detection categories was 12:20:48 (hh:mm:ss), while Mask R-CNN took 24:26:10 (hh:mm:ss). The joint detection of all the detection categories reduced the overall training time in MAD-Net by approximately 50% compared with Mask R-CNN. As expected, YOLOv5s-Seg model had the fastest total training time of 08:38:27 (hh:mm:ss); however, the accuracy in the binary image segmentation was significantly lower than for the rest of the models (Mask R-CNN, MAD-Net, YOLACT). The resulting bounding boxes from YOLOv5s-Seg model did have a comparable bounding box mAP score to the other tested models, however. The YOLACT model generally had the most consistent mAP scores across all object classes from all detection categories. The MAD-Net and Mask R-CNN models also had consistent mAP scores on the validation sets and the Test-S set; however, the mAP scores on the Test-I set were significantly less in MAD-Net and Mask R-CNN, since both models were unable to detect instances from different classes that were exactly overlapping each other.

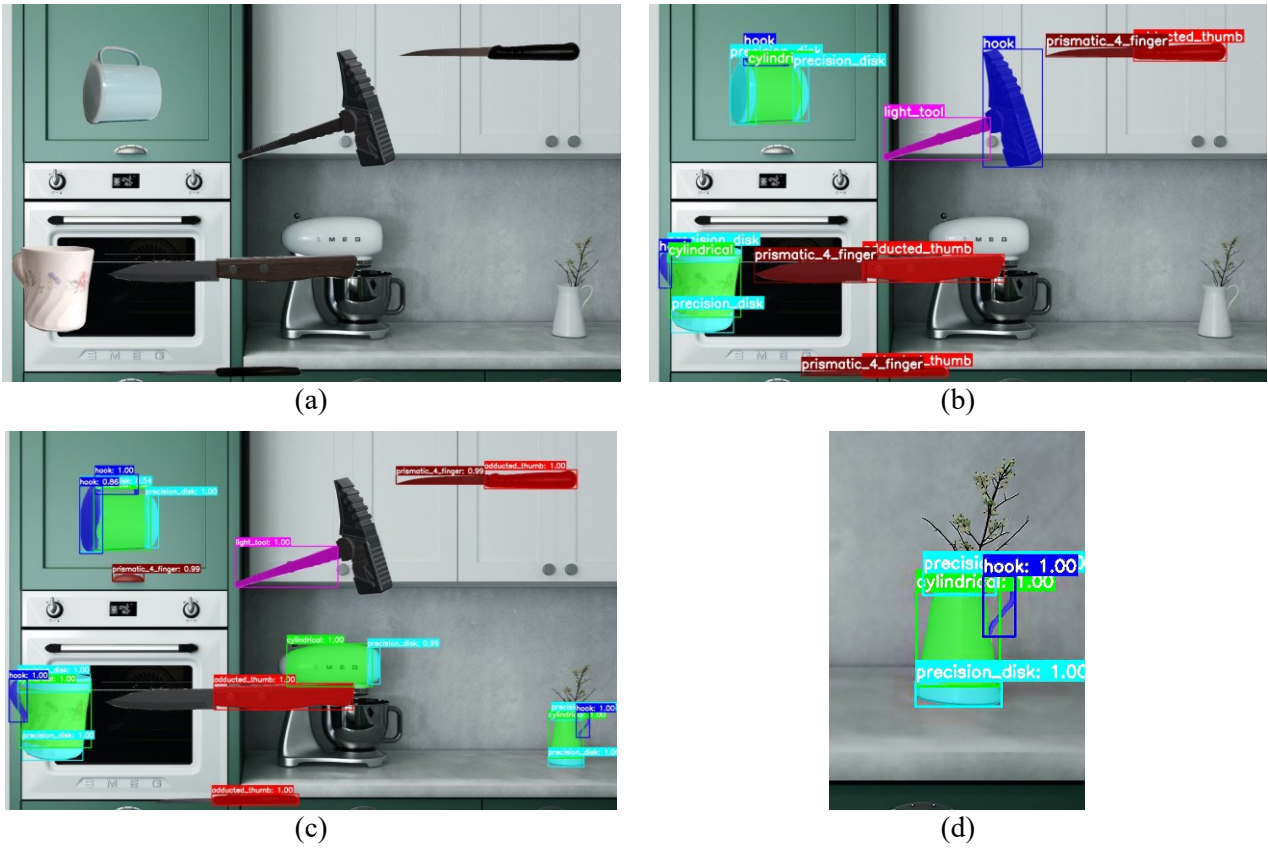
In the MAD-Net and Mask R-CNN model, the failure to detect overlapping instances unlike YOLACT and YOLOv5s-Seg was possibly due to the initial NMS filtering of the ROI proposals. Before an object

class was assigned at the end of the RPN, the predicted ROI boxes were first filtered with NMS. At that stage, the “object class” was separated between each feature map level, where each value of the feature map was an indicator for objectness. Therefore, the ROI boxes with a similar same aspect ratio and dimensions were filtered out, which means that the possibility of duplicate instances were nearly impossible. The annotated task regions in the MOMA dataset contained excessively cluttered and overlapping instances, especially between the *carry* and the *use* class. In real life applications, the *use* task would constitute a specific task on a specific object class. For example, a *cup* can be *used* to pour or drink liquids, while a *bottle* can be used to pour, drink, and dispense liquids. With the further denominations of the task classes, the duplicate instance problem can be solved. However, due to the limitations of training a supervised learning model, any new object classes (tasks in this case) are not understandable by the trained model since it was not trained on related examples. Therefore, the probability of a trained supervised model on an object with new object classes would be zero. One method to solve this problem is to introduce a KB graph, whereby similar tasks can be correlated to one or more of the annotated grasping locations. For all models, the difference in mAP scores between the Test-S and Test-I sets was mainly due to the inconsistent classifications of the instance regions during annotation. For example, the *hammer* object was expected to have a *hook* grasp type at the tool head. However, a mallet, which was also classified as a *hammer* in this case, had a cylindrical tool head, which is only suitably grasped with the *cylindrical* grasp type.

The poor performance of YOLOv5s-Seg’s ability to segment image mask was mostly due to the condensed detection of the bounding boxes, object classes, and binary image masks under a single prediction head for each anchor scale. The prediction head in YOLOv5s-Seg was a single convolutional layer that contained the channels for the objectness score, object class logits, bounding box coordinates, and the mask coefficients used to assemble the final binary image masks. As seen in YOLACT, Mask R-CNN, and MAD-Net, each prediction head was decoupled to perform a single task: to predict the bounding boxes with its corresponding object class, and independently segment a binary image mask.

Sometimes, the detection of the instances in an RGB scene may have had a positive prospect in a model’s overall performance for unseen objects. As seen in Fig. 4.14c, the *cup* object located at the bottom right of the image (or Fig. 4.14d) should have been a false positive. The object was supposed to be a jug that was currently used to hold live plants; therefore, the jug was technically classified as a vase. The discrepancies in the classification scheme of the objects were a significant issue during annotation, training, and evaluation. However, in the case of Fig. 4.14, despite that classifying the vase as a cup was

technically incorrect, the localization and segmentation of the object was very accurate, and the potential grasp-type regions were labelled correctly. Therefore, based on the predictions in Fig. 4.14c, this particular model (MAD-Net) had generalized to unseen objects.



**Figure 4.14:** Example scenario where a model incorrectly detected an object but provided the correct grasp-type affordances for the unaccounted (previously unseen) object: (a) original image, (b) ground truth, (c) detection of grasp-type instances by MAD-Net, and (d) enlarged location of the detected anomaly.



## Chapter 5

### Conclusion

The recently, commercially available myoelectric prosthetic hands [17, 18, 16, 19] use sEMG pattern recognition to simultaneously control the movement of the prosthetic arm, select a grasp type, and identify the action intent to perform a grasping action. Therefore, the coupled control strategy in myoelectric prostheses will increase the inaccuracies in predicting the intended grasp type for a grasping task due to the possibility of misclassifying similar sEMG patterns for other actions on the prosthetic arm. For example, an sEMG signal can be mistaken for other grasp types or the movements of the arm that the prosthetic user did not intend to perform. To decouple the control strategy of a myoelectric prosthetic hand, recent vision-based current applications in prosthetic grasping use a camera-based vision system to predict a grasp type from an object, before the sEMG signals from the residual limb trigger the action intent of grasping the selected object. However, current vision-based methods are only limited to the detections of a single grasp type for a single object (cups, bottles, etc.) with a simple CNN classification network [29, 25, 27]. In robotic applications, the detection of the object affordances [8, 9, 10] and grasp affordances [11, 6, 5, 7, 12] allows the inference of a task, in the same manner as how humans semantically understand the relationships between an object and their surroundings. Since the current vision-based prostheses are limited to the selection of stable grasps over task-suitable grasps, the detection of grasp affordances was adapted from the robotic grasping applications to complete the thesis objectives. The grasp affordances are the possible regions on an object that affords to be grasped for a specific task [11, 6, 5, 7, 12].

In application to prosthetic grasping, the main objective of this thesis research was to develop a vision-based, deep-learning model that can simultaneously detect all objects and their grasp affordances in each RGB multi-object scene. Based on the outcome of the ablation study in Sec. 3.1, the localization of an object in an RGB image was necessary to precisely localize the objects and their grasp affordances to segment the binary image masks, as seen in the AffordanceNet [8] model. The image segmentation of the grasp affordances also provided the exact shape and assists the implicit classification of an object part, which can ultimately influence the type of grasp type and task that can be performed in the object region. Unfortunately, the IIT-AFF [8] dataset used to train the AffordanceNet [8] model only contains the image masks of the object affordances, or the actions that an object can provide to a user, along with the bounding boxes of the objects. Unlike the instance segmentation models trained in this thesis research,

AffordanceNet [8] is also only capable of assigning a single image pixel location to an affordance class, or the grasp affordances in this case. Therefore, to complete the main thesis objectives, the MOMA synthetic dataset was first created and annotated, where the existing Mask R-CNN [33], YOLACT [82], and YOLOv5-Seg [83] instance segmentation models were initially trained and evaluated on. Secondly, a new joint detection model called MAD-Net was then developed in this thesis research (Sec. 3.4.4) to make possible the simultaneous detection of objects, grasp types, and task regions as instances.

As seen in Chapter 4, the developed MAD-Net model outperformed Mask R-CNN [33], YOLACT [82], and YOLOv5-Seg [83] in terms of the overall average mAP scores and the quality of the produced inferencing images. In comparison with Mask R-CNN, the base model that MAD-Net was based on, MAD-Net outperforms Mask R-CNN in all performance categories, including the overall training time, all mAP scores across all detection categories (objects, grasp types, and task regions), and the GPU usage during training. MAD-Net only requires a small overhead of 2.05 GB to simultaneously train over all detection categories, while requiring approximately a 50% reduced training time to do so. Unfortunately, both MAD-Net and Mask R-CNN suffer from an overall mAP score decrease on the images of unseen unique objects (Test-I). The reduced performance of MAD-Net and Mask R-CNN was largely due to the NMS filtering of the ROI regions at the end of the RPN, since the duplicate box proposals were removed. Although the mAP score was a significant metric in the determination of how well a model has performed, a false positive in the detection of instances may not always be unfortunate. As discussed in Sec. 4.3, some background objects are misclassified due to the limitations of a supervised learning network, such as the detection of an instance on new unseen object class. The general object class is mostly insignificant since the related object with a different class may afford similar tasks and grasp types. The object image mask helps localize and bound the regions of the object parts, which influences the increase in mAP accuracy, as seen in the MAD-Net model.

Overall, the joint detection of the objects and their grasp affordances as the instances of objects, grasp types, and task regions is feasible for vision-based prostheses. While the developed vision system produces a selection of possible grasp types on a single object, current commercially available, myoelectric prostheses can still handle a small selection of up to eight different pre-programmed grasp types at a time [17, 18, 16, 19]. In most cases, the determination of the object class and the possible task classes filter the number of possible grasp types available to the prosthetic user.

## 5.1 Future Work and Improvements

The current vision system developed and implemented in this thesis research can still be improved for future research. While an object is no longer limited to a single grasp type, the vision system is still lacking a task recognition system. Currently, only one grasp type was associated with the object part, which was the intended design. However, as seen in Chapter 2, a task requiring more stability or dexterity may change the suggested grasp type on the object part. For example, the *light tool* grasp type can be used to hold the hammer, only when a nail is driven into a piece of wood. However, if the user is casually carrying the hammer around, the *light tool* grasp type may not be preferred over the more stable *cylindrical* grasp type. The current task-region detection method does filter some grasp-type regions based on the possible task. However, the task regions are only detected, and the tasks are currently not inferred.

To solve the issue with the contested grasp types on an object part due to the changing task, one method could involve the creation of a grasp database. The grasp database can include the common contact points on an object, such as in the database ContactDB [87], for the task class to infer a grasp type. The benefit of this new approach is that the new object, grasp type, and task classes can be automatically and continuously updated in the database. Therefore, the manual and semi-automatic annotation methods used in Chapter 3 can be eliminated, as training examples can be generated by the participating users. In addition, the suitable tasks no longer need to be determined as an instance mask and bounding box. Another benefit of using a database to store grasp types, is that the participating users could use a simulated prosthesis to provide the training examples; alternatively, the same grasp database could be available to the prosthetic user when they need a new prosthesis replacement. In an alternative method, gaze estimation [88, 89, 90, 91] can be implemented to assist the inference of the available tasks. Gaze estimation [88, 89, 90, 91] predicts what the user is looking at, which can highly suggest the type of task they intend to perform on what object. Once the task and object are selected, suitable grasp types can be inferred by mapping the task regions to the grasp-type regions that were detected from a grasp affordance detector like MAD-Net. Alternatively, the suitable grasp types can be inferred from a grasp database that has the matching task and object.

Currently, the MAD-Net model is not optimized for real-time applications, which is necessary for prosthetic grasping applications. For the instance segmentation of objects (e.g. cups, bottles), Mask R-CNN [33] has an inference speed of 5 frames per second (fps), whereas YOLACT [82] and YOLOv5-Seg

[83] could achieve more than 30 fps. Since MAD-Net is more computational expensive than the base model, Mask R-CNN, the inference speed of the MAD-Net model would most likely be less than 5 fps. Therefore, MAD-Net may not be suitable for real-time applications. Fortunately, the joint detection of objects and their grasp affordances could still be applied to the YOLACT and YOLOv5-Seg models for faster inference speeds and real-time processing of RGB images.

## References

- [1] A. Bernardino, M. Henriques, N. Hendrich and J. Zhang, "Precision grasp synergies for dexterous robotic hands," *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 62-67, 2013.
- [2] G. Vasan and P. M. Pilarski, "Learning from demonstration: Teaching a myoelectric prosthesis with an intact limb via reinforcement learning," *2017 International Conference on Rehabilitation Robotics (ICORR)*, pp. 1457-1464, 2017.
- [3] M. Cai, K. M. Kitani and Y. Sato, "Understanding Hand-Object Manipulation with Grasp Types and Object Attributes," *Proceedings of Robotics: Science and Systems*, vol. 3, 2016.
- [4] J. J. Gibson, "The Theory of Affordances," in *The Ecological Approach To Visual Perception*, New York and London, Psychology Press, Taylor & Francis Group, 2014, pp. 119-135.
- [5] E. Corona, A. Pumarola, G. Alenyà, F. Moreno-Noguer and G. Rogez, "GanHand: Predicting Human Grasp Affordances in Multi-Object Scenes," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5030-5040, 2020.
- [6] R. Detry, J. Papon and L. Matthies, "Task-oriented grasping with semantic and geometric scene understanding," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3266-3273, 2017.
- [7] P. Ardón, . È. Pairet, R. Petrick, S. Ramamoorthy and K. Lohan, "Reasoning on Grasp-Action Affordances," *Towards Autonomous Robotic Systems. TAROS 2019*, vol. 11649, pp. 3-15, 2019.
- [8] T.-T. Do, A. Nguyen and I. Reid, "AffordanceNet: An End-to-End Deep Learning Approach for Object Affordance Detection," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5882-5889, 2018.
- [9] S. Deng, X. Xu, C. Wu, K. Chen and K. Jia, "3D AffordanceNet: A Benchmark for Visual Object Affordance Understanding," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1778-1787, 2021.

- [10] M. Kovic, J. A. Stork, J. A. Haustein and D. Kragic, "Affordance detection for task-specific grasping using deep learning," *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 91-98, 2017.
- [11] D. Song, C. H. Ek, K. Huebner and D. Kragic, "Task-Based Robot Grasp Planning Using Probabilistic Inference," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 546-561, 2015.
- [12] P. Ardón, È. Pairet, R. P. A. Petrick, S. Ramamoorthy and K. S. Lohan, "Learning Grasp Affordance Reasoning Through Semantic Relations," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4571-4578, 2019.
- [13] A. L. Muilenburg and M. A. LeBlanc, "Body-Powered Upper-Limb Components," in *Comprehensive Management of the Upper-Limb Amputee*, New York, Springer, 1989, pp. 28-38.
- [14] Fillauer, "Hosmer #88X Hook," [Online]. Available: <https://fillauer.com/products/hosmer-88x-hook/>. [Accessed 19 07 2023].
- [15] Naked Prosthetics, "MCPDriver, Instruction For Use, Body-Driven, Class 1 Prosthetic Device," 2021. [Online]. Available: <https://www.npdevices.com/product/mcpdriver/>. [Accessed 19 07 2023].
- [16] Open Bionics, "HERO ARM User Manual," 2022. [Online]. Available: <https://openbionics.com/en/hero-arm-user-guide/>. [Accessed 19 07 2023].
- [17] Ottobock, "Axon-Bus Prosthetic System - Instructions For Use (User)," 2021. [Online]. Available: <https://shop.ottobock.us/Prosthetics/Upper-Limb-Prosthetics/Michelangelo-Axon-Bus-System/Michelangelo-Hand-AxonHook/Axon-Bus%C2%AE-Prosthetic-System/p/8K5000>. [Accessed 19 07 2023].
- [18] Ottobock, "bebionic Hand Instructions for Use," 2022. [Online]. Available: <https://shop.ottobock.us/Prosthetics/Upper-Limb-Prosthetics/bebionic/c/2888>. [Accessed 19 07 2023].
- [19] Össur, "Instructions For Use, i - LIMB HAND," 2021. [Online]. Available: <https://www.ossur.com/en-ca/prosthetics/arms/i-limb-quantum>. [Accessed 19 07 2023].

- [20] J. B. Bennett and C. B. Alexander, "Amputation Levels and Surgical," in *Comprehensive Management of the Upper-Limb Amputee*, New York, Springer, 1989, pp. 1-10.
- [21] M. R. Cutkosky, "On grasp choice, grasp models, and the design of hands for manufacturing tasks," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 3, pp. 269-279, 1989.
- [22] T. Feix, J. Romero, H.-B. Schmiedmayer, A. M. Dollar and D. Kragic, "The GRASP Taxonomy of Human Grasp Types," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 66-77, 2016.
- [23] A. Camacho-Ramirez, Á.-V. C. Juan, B. Saldivar, A. H. Vilchis-González and J. M. Jacinto-Villegas, "Adjustable Stiffness-Based Supination–Pronation Forearm Physical Rehabilitator," *The Present and Future of Robotic Technology in Rehabilitation*, vol. 12, no. 12, 2022.
- [24] S. Hussain, S. Shams and S. J. Khan, "Impact of Medical Advancement: Prostheses," *Computer Architecture in Industrial, Biomechanical and Biomedical Engineering*, 2019.
- [25] G. Ghazaei, A. Alameer, P. Degenaar, G. Morgan and K. Nazarpour, "Deep learning-based artificial vision for grasp classification in myoelectric hands," *Journal of Neural Engineering*, vol. 14, p. 036025, 2017.
- [26] C. Shi, D. Yang, J. Zhao and H. Liu, "Computer Vision-Based Grasp Pattern Recognition With Application to Myoelectric Control of Dexterous Hand Prosthesis," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 9, pp. 2090-2099, 2020.
- [27] D. T. Andrade, A. Ishikawa, A. D. Munoz and E. Rohmer, "A Hybrid Approach for the Actuation of Upper Limb Prostheses Based on Computer Vision," *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, pp. 1-6, 2017.
- [28] L. C. Smail, C. Neal, C. Wilkins and T. L. Packham, "Comfort and function remain key factors in upper limb prosthetic abandonment findings of a scoping review," *Disability and Rehabilitation: Assistive Technology*, vol. 16, no. 8, pp. 821-830, 2020.

- [29] A. Asghar, "Fusion of Estimated Depth and RGB Features for Improved Grasp-Type Selection of Novel Objects," *UWSpace*, Vols. MASc thesis, Dept. Systems Design Eng., Waterloo Univ., Waterloo, ON, CA., 2022.
- [30] F. Vasile, E. Maiettini, G. Pasquale, A. Florio, N. Boccardo and L. Natale, "Grasp Pre-shape Selection by Synthetic Training: Eye-in-hand Shared Control on the Hannes Prosthesis," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13112-13119, 2022.
- [31] A. J. Spiers, J. Cochran, L. Resnik and A. M. Dollar, "Quantifying Prosthetic and Intact Limb Use in Upper Limb Amputees via Egocentric Video: An Unsupervised, At-Home Study," *IEEE Transactions on Medical Robotics and Bionics*, vol. 3, no. 2, pp. 463-484, 2021.
- [32] A. J. Spiers, L. Resnik and A. M. Dollar, "Analyzing at-home prosthesis use in unilateral upper-limb amputees to inform treatment & device design," *2017 International Conference on Rehabilitation Robotics (ICORR)*, pp. 1273-1280, 2017.
- [33] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," *IEEE International Conference on Computer Vision (ICCV)*, pp. 2980-2988, 2017.
- [34] C. Ferrari and J. Canny, "Planning optimal grasps," *Proceedings 1992 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2290-2295, 1992.
- [35] C. Borst, M. Fischer and G. Hirzinger, "Grasp planning: how to choose a suitable task wrench space," *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, pp. 319-325, 2004.
- [36] D.-A. Huang, M. Ma, W.-C. Ma and K. M. Kitani, "How do we use our hands? Discovering a diverse set of common grasps," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 666-675, 2015.
- [37] M. Cai, K. M. Kitani and Y. Sato, "A scalable approach for understanding the visual structures of hand grasps," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1360-1366, 2015.



- [38] M. Cai, K. M. Kitani and Y. Sato, "An Ego-Vision System for Hand Grasp Analysis," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 4, pp. 524-535, 2017.
- [39] F. Zhang *et al.*, "MediaPipe Hands, On-device Real-time Hand Tracking," *arXiv preprint arXiv:2006.10214*, 2020.
- [40] G. Garcia-Hernando, S. Yuan, S. Baek and T.-K. Kim, "First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 409-419, 2018.
- [41] S. Yuan, Q. Ye, B. Stenger, S. Jain and T.-K. Kim, "BigHand2.2M Benchmark: Hand Pose Dataset and State of the Art Analysis," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2605-2613, 2017.
- [42] Y. Yang, C. Fermüller, Y. Li and Y. Aloimonos, "Grasp type revisited: A modern perspective on a classical feature for vision," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 400-408, 2015.
- [43] W. Shang, F. Song, Z. Zhao, H. Gao, S. Cong and Z. Li, "Deep Learning Method for Grasping Novel Objects Using Dexterous Hands," *IEEE Transactions on Cybernetics*, vol. 52, no. 5, pp. 2750-2762, 2022.
- [44] K. Karunratanakul, J. Yang, Y. Zhang, M. J. Black, K. Muandet and S. Tang, "Grasping Field: Learning Implicit Representations for Human Grasps," *2020 International Conference on 3D Vision (3DV)*, pp. 333-344, 2020.
- [45] C. Gabellieri *et al.*, "Grasp It Like a Pro: Grasp of Unknown Objects With Robotic Hands Based on Skilled Human Expertise," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2808-2815, 2020.
- [46] C. Yang, X. Lan, H. Zhang and N. Zheng, "Task-oriented Grasping in Object Stacking Scenes with CRF-based Semantic Model," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6427-6434, 2019.

- [47] E. Ragusa, C. Gianoglio, R. Zunino and P. Gastaldo, "Data-Driven Video Grasping Classification for Low-Power Embedded System," *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 871-874, 2020.
- [48] S. Narasimhaswamy, Z. Wei, Y. Wang, J. Zhang and M. H. Nguyen, "Contextual Attention for Hand Detection in the Wild," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9566-9575, 2019.
- [49] M. Kovic, D. Kragic and J. Bohg, "Learning Task-Oriented Grasping From Human Activity Datasets," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3352-3359, 2020.
- [50] M. Kovic, D. Kragic and J. Bohg, "Learning to Estimate Pose and Shape of Hand-Held Objects from RGB Images," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3980-3987, 2020.
- [51] F. Quivira, T. Koike-Akino, Y. Wang and D. Erdogmus, "Translating sEMG signals to continuous hand poses using recurrent neural networks," *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pp. 166-169, 2018.
- [52] Y. Liu, S. Zhang and M. Gowda, "NeuroPose: 3D Hand Pose Tracking using EMG Wearables," *Proceedings of the Web Conference 2021*, pp. 1471-1482, 2021.
- [53] C. D. Santina *et al.*, "Learning From Humans How to Grasp: A Data-Driven Architecture for Autonomous Grasping With Anthropomorphic Soft Hands," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1533-1540, 2019.
- [54] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779-788, 2016.
- [55] W. Liu *et al.*, "SSD, Single Shot MultiBox Detector," *Computer Vision – ECCV 2016*, vol. 9905, pp. 21-37, 2016.

- [56] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2016.
- [57] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell and M. Aubry, "A Papier-Mache Approach to Learning 3D Surface Generation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 216-224, 2018.
- [58] J. Romero, D. Tzionas and M. J. Black, "Embodied hands: modeling and capturing hands and bodies together," *ACM Transactions on Graphics*, vol. 36, no. 6, pp. 1-17, 2017.
- [59] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149-2154, 2004.
- [60] A. T. Miller, S. Knoop, H. I. Christensen and P. K. Allen, "Automatic grasp planning using shape primitives," *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, pp. 1824-1829, 2003.
- [61] R. Balasubramanian, L. Xu, P. D. Brook, J. R. Smith and Y. Matsuoka, "Physical Human Interactive Guidance: Identifying Grasping Principles From Human-Planned Grasps," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 899-910, 2012.
- [62] O. Glauser, S. Wu, P. Daniele, H. Otmar and O. Sorkine-Hornung, "Interactive hand pose estimation using a stretch-sensing soft glove," *ACM Transactions on Graphics*, vol. 38, no. 4, pp. 1-15, 2019.
- [63] J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431-3440, 2015.
- [64] R. Meyes, M. Lu, C. Waubert de Puiseau and T. Meisen, "Ablation Studies in Artificial Neural Networks," 2019.
- [65] T. -Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *Computer Vision – ECCV 2014*, vol. 8693, no. Lecture Notes in Computer Science, pp. 740-755, 2014.

- [66] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 2016.
- [67] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211-252, 2015.
- [68] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems 32*, 2019.
- [69] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Proceedings of the 3rd International Conference for Learning Representations (ICLR 2015)*, 2015.
- [70] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision (IJCV)*, vol. 88, pp. 303-338, 2010.
- [71] D. Dwibedi, I. Misra and M. Hebert, "Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1310-1319, 2017.
- [72] Luxonis, "Depth AI: Embedded Machine learning and Computer vision api," 2020. [Online]. Available: <https://luxonis.com/>. [Accessed 15 November 2023].
- [73] Luxonis, "OAK-D: Stereo camera with Edge AI," 2020. [Online]. Available: <https://luxonis.com/>. [Accessed 15 November 2023].
- [74] D. McCurdy, "three-gltf-viewer," 2 February 2020. [Online]. Available: <https://github.com/donmccurdy/three-gltf-viewer>. [Accessed 15 11 2023].
- [75] B. Dwyer *et al.*, "Roboflow (Version 1.0)," 2022. [Online]. Available: <https://roboflow.com>. [Accessed 15 November 2023].
- [76] K. Wada, "Labelme: Image Polygonal Annotation with Python," 22 August 2023. [Online]. Available: <https://github.com/wkentaro/labelme>. [Accessed 18 November 2023].

- [77] G. Bradski, "Dr. Dobb's Journal of Software Tools," 15 January 2008. [Online]. Available: <https://github.com/opencv>. [Accessed 18 November 2023].
- [78] E. Behar and J.-M. Lien, "Dynamic Minkowski sum of convex shapes," *2011 IEEE International Conference on Robotics and Automation*, pp. 3463-3468, 2011.
- [79] D. H. Douglas and T. K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature," in *Classics in Cartography: Reflections on Influential Articles from Cartographica*, Chichester, John Wiley & Sons, Ltd, 2011, pp. 15-28.
- [80] G. Tanner, "Synthetic Data Generation for Object Detection and Instance Segmentation," 21 December 2020. [Online]. Available: <https://github.com/TannerGilbert/Computer-Vision-Synthetic-Data-Generation>. [Accessed 18 November 2023].
- [81] S. Gillies, C. Van der Wel, J. Van den Bossche, M. W. Taves, J. Arnott and B. C. Ward, "Shapely: Manipulation and analysis of geometric objects in the Cartesian plane.," 12 October 2023. [Online]. Available: <https://github.com/shapely/shapely>. [Accessed 18 November 2023].
- [82] D. Bolya, C. Zhou, F. Xiao and Y. J. Lee, "YOLACT: Real-Time Instance Segmentation," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9156-9165, 2019.
- [83] G. Jocher *et al.*, "ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation (v7.0)," Ultralytics, 2022.
- [84] T. -Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936-944, 2017.
- [85] C. -Y. Wang, H. -Y. Mark Liao, Y. -H. Wu, P. -Y. Chen, J. -W. Hsieh and I. -H. Yeh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1571-1580, 2020.
- [86] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye and D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 7, 2020.

- [87] S. Brahmabhatt, C. Ham, C. C. Kemp and J. Hays, "ContactDB: Analyzing and Predicting Grasp Contact via Thermal Imaging," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8701-8711, 2019.
- [88] Y. Li, A. Fathi and J. M. Rehg, "Learning to Predict Gaze in Egocentric Video," *2013 IEEE International Conference on Computer Vision*, pp. 3216-3223, 2013.
- [89] Y. Li, M. Liu and J. M. Rehg, "In the Eye of the Beholder: Gaze and Actions in First Person Video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 6731-6747, 2023.
- [90] H. Kim, Y. Ohmura and Y. Kuniyoshi, "Using Human Gaze to Improve Robustness Against Irrelevant Objects in Robot Manipulation Tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4415-4422, 2020.
- [91] Y Huang, M Cai, Z Li, Y Sato, "Predicting Gaze in Egocentric Video by Learning Task-dependent Attention Transition," *Proceedings of the European Conference on Computer Vision (ECCV), 2018*, pp. 754-769, 2018.
- [92] M. Everingham and J. Winn, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit," *Pattern Analysis, Statistical Modelling and Computational Learning*, 2012.