SINGLE SHOT MULTI BOX DETECTOR APPROACH TO AUTONOMOUS VISION-BASED PICK AND PLACE ROBOTIC ARM IN THE PRESENCE OF UNCERTAINTIES

PATRICK KIPKOSGEI CHEMELIL

MASTER OF SCIENCE
(MECHATRONIC ENGINEERING)

JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

2021

Single Shot Multi Box Detector Approach to Autonomous Vision-Based Pick and Place Robotic Arm in The Presence of Uncertainties

Patrick Kipkosgei Chemelil

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Mechatronic Engineering of the Jomo Kenyatta University of Agriculture and Technology

2021

DECLARATION

This thesis is my original work and has not been presented for a degree in any other university.

Signature..... Date...../...../.....

Patrick Kipkosgei Chemelil

This thesis has been submitted for examination with our approval as university supervisors:

Signature..... Date...../...../.....

Dr.-Ing. Jackson G. Njiri, PhD

JKUAT, Kenya

Signature..... Date...../...../.....

Dr.-Ing. James K. Kimotho, PhD

JKUAT, Kenya

DEDICATION

I dedicate this work to my parents Mr and Mrs, Cherogony, my friends, and colleagues who have supported me throughout. Thank you all for the support during my research journey.

ACKNOWLEDGEMENTS

Thanks be to God for the gift of life, good health and for sustaining me throughout this research. I also thank my supervisors Dr.-Ing. Jackson G. Njiri and Dr.-Ing. James K. Kimotho for their invaluable guidance and advice throughout my studies.

Special thanks goes to Jomo Kenyatta University of Agriculture and Technology (JKUAT) for granting me the scholarship for my studies and for providing additional funding for this research. My appreciation also goes to the staff of JKUAT for the assistance they accorded me. I also thank my course-mates, who walked this journey with me, for their encouragement and invaluable support.

I want also to extend my gratitude to the technologists and the staff members of the department of Mechatronic Engineering for their assistance and support during this period. My sincere gratitude also goes to my family who gave me moral support and guidance through out my journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CNN	Convolutional Neural Network
DH	Denavit-Hartenberg
DOF	Degrees of Freedom
DPM	Discriminatively trained Part-based Model
FSR	Force Sensing Resistor
GB	Giga Byte
GCMLE	Google CLoud Machine Learning Engine
IOU	Intersection Over Union
PID	Proportional Integral Derivative
RCNN	Region-based Convolutional Neural Network
ResNet	Residual Neural Network
RGB	Red Green Blue
SSD	Single Shot multibox Detector
TPU	Tensor Processing Unit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VGG	Visual Geometry Group neural network
VM	Virtual Machine

LIST OF SYMBOLS

fps	Frames Per Second
lm	lumens
mAP	mean Average Precision
x	x-axis coordinate
y	y-axis coordinate
z	z-axis coordinate

ABSTRACT

This research presents a problem of real-time accurate object detection in picking and placing objects using a robotic arm in conditions where conventional appearance-based approaches are largely ineffective. These conditions include partial occlusion, varying lighting, and change in camera pose. The methods presented in the literature have managed to achieve real time detection but at the expense of presence of the above mentioned scenarios which make them not usable in real world application. A single shot multibox detector Convolutional Neural Network is proposed to handle this problem. The network has been used for object detection in robotics but the performance of SSD with Resnet-50 as the backbone has not been explored. To evaluate the performance of the network, some challenges were formulated. The network was tasked to identify objects under uncertain conditions of varied lighting, partial occlusion, and changing camera pose. This was achieved by using bulbs of different lumen, occluding the objects in a manner that half of the object was visible to the camera and viewing the objects at an angle of 45°. This angle was different from the training viewing angle that was 0°. The network's performance and speed of detection was tabulated for every experiment. The robot's performance with the network was then evaluated by timing how long it took to identify, pick objects from one location, and place them in another. Successful attempts at grasping the objects were also evaluated. The proposed network helps to achieve real time detection in the range of 40 frames per second (fps) with accuracies of above 0.69 mAP (mean average precision) in varied lighting conditions, partial occlusion, and changing camera pose. This is an improvement to the SSD300 which was using VGG16 and produced 30 fps with accuracies of 0.65 mAP. Autonomous pick and place function was tested and was found to take between 15-30 seconds. The time was a factor of the shape of the object to be detected and how easy it was to pick and place. Experimental results validated the performance of the network and robot control method in a realistic scenario of picking and placing objects.

CHAPTER ONE

INTRODUCTION

1.1 Background

Use of robots on the production floor is gaining popularity due to low robot labour cost per hour of $2-3 as compared to the steadily rising human cost of $25 per hour (Karabegović, Karabegović, Mahmić, & Husak, 2015). The global demand for products has increased significantly due to e-commerce and platforms like Alibaba, Ebay, and Amazon. These platforms have leveraged on the increased internet penetration to sell goods all over the world which has driven companies to increase their production levels to meet the demand (Javadi, Dolatabadi, Nourbakhsh, Poursaeedi, & Asadollahi, 2012).

Quality control has also driven the adoption of pick and place robots in factories, especially in fruit industry. Such an industry relies a lot on skilled labourers who have years of experience to pick the right fruit. When fitted with a camera, a robot can pick the best fruit quality to be processed far better than a skilled labourer (Kondo, 2010).

Controlling such robots has been done predominately by proportional integral derivative (P.I.D.) (Ferdinando, Wicaksono, & Wibowo, 2017) and robust controllers (Houska, Li, & Chachuat, 2018). In this case, the robots is fitted with a proximity sensor and a motorized arm is used to pick the desired object. The controllers are employed to sequence the motion of the arm and fingers after the sensor identified an object. These robots reduce production time and labour costs but visual inspection still relies on skilled personnel. There is need to give the visual functionality to robots to increase productivity, reduce cost while maintaining or improving quality (Gu, Xiong, & Wan, 2013).

Vision-based control has shown a lot of potential due to the advancement of high quality imaging devices and accelerated development in image processing technology. Object detection and robotics go hand in hand especially in visual robotic control and has been a rich area of research (Kumar, Lal, Kumar, & Chand, 2014; H. I. Lin, Chen, & Chen, 2015; Viola & Jones, 2011). These techniques have had short comings in real-time object detection or have low accuracies when faced with uncertainties like occlusion or varying light intensity. Brachmann et al (Brachmann et al., 2014) used a learned object representation

combining dense 3D object coordinate labeling and dense class labelling for textured and texture-less objects. These modern techniques are more accurate with object identification and are also close to achieving real time detection.

Neural Networks pioneered by Hubel (Akogo & Palmer, 2019) has shown promise in image processing and robot vision control. Lecun used a model of neural network that had more neurons to develop a network called LeNet5 (Yoo, 2015). LeNet5 was able to achieve character recognition of handwritten digits using a small memory footprint as compared to pre-existing methods. With these achievements, LeNet5 was the foundation for other networks and their accuracy levels kept increasing constantly.

The next major milestone in neural network based image detection is AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) in 2012. It was the first network to use several convoluted layers to achieve image detection and it ushered in the age of deep convolutional neural networks. Since then, deep convolutional neural networks have been used in robotics by Ahlin et al (Konrad Ahlin & McMurray, 2016), Zeng et al (et al, 2017), De Magidtris et al (De Magistris, Munawar, & Vinayavekhin, 2016), and Hossain et al (Hossain & Capi, 2016). Though there is significant development in vision-based pick and place robotics, there exists some limitation which are highlighted in the next section.

1.2 Problem Statement

Vision-based pick and place robots have become part of many industries because of their ability to perform tedious and repetitive work more accurately without fatigue and at a lower cost. Several techniques such as Stereo vision system have been implemented to continuously improve visual robots and of late convolutional neural networks (CNN) have become the most preferred method for object detection.

These CNNs have very high image classification accuracies and even rival human beings in recognising objects in images. They have also been adapted to recognize objects and give their location in an image with high accuracies. Despite the high accuracies, CNNs still have challenges with some conditions like variation of light intensity, partial occlusion, different object pose and viewing angle. Attaining real time speeds when performing object detection is also an obstacle for CNNs that limits it from being used in high accuracy high speed applications like pick and place robotics.

This research proposes a vision based convolutional neural network control system for a pick and place robot that can achieve real-time speeds (30 frames per second) despite uncertainties like variation of light intensity, partial occlusion, different object pose and viewing angle.

1.3 Objectives

The goal of this work is to develop a convolutional neural network to detect objects using a vision sensor and achieve real time speeds under uncertainty conditions. These conditions of variation of light intensity, occlusion, and viewing angle should be overcome while performing pick and place functions using a robotic arm. To achieve the main objective, the following specific objectives were realized.

1. To develop a CNN motion control algorithm and a grasping algorithm for a pick-and-place robot.

2. To implement the CNN in a pick and place robot and evaluate its functionality.

3. To evaluate the performance of the entire system (object detection and robotic arm) in terms of accuracy of the overall system in performing pick-and-place tasks through experiments.

1.4 Scope

The scope of this research will be to develop a Convolutional Neural Network, evaluate the performance of the CNN under uncertain conditions of varying light intensity, partial occlusion and viewing angle and, test pick-and-place robotic arm functionality. For evaluation of the CNN, 12 object classes will be used and three sets of grasping attempts will be carried to explore the accuracy of the pick-and-place system.

1.5 Justification

In robotics, pick and place robotic arms have become the focus of warehousing and manufacturing due to their accuracies and high productivity as compared to human beings. With the increased use of these machines, more challenges are experienced which need robust solutions. For robotic arms that are used to identify objects, pick and place them,

accuracy of detection and speed has become an important factor.

Developing the control systems that achieve object detection, picking and placing objects will mean an increase in productivity. In the long term, reduction in the cost of production will result in lower prices which benefits the consumers. For the companies that utilise this technology, the increase in productivity means a reduction in operating costs hence increased profits. For the researchers, this provides an opportunity to further research into this field and fill gaps in research.

1.6 Outline of the Thesis

This thesis contains five chapters. The current chapter is the introduction to the research which presents a general overview of the existing problem related to object detection in pick and place robots and other shortcomings. A review of the existing literature on research that has been done regarding object detection and robot kinematics is presented in Chapter 2.

Experimental setup and the measured parameters to determine performance of the proposed network and overall system is presented in Chapter 3. Results of the network training, performance under varied conditions and picking and placing are presented and discussed in Chapter 4. Conclusions and recommendations are presented in Chapter 5.

CHAPTER TWO

LITERATURE REVIEW

2.1 Object Detection Overview

Object detection involves identifying an object in an image and locating its position. Object detection is hence divided into three steps: information region selection, feature extraction, and classification (Zhao, Zheng, Xu, & Wu, 2019).

Information region selection: Objects may have different sizes, aspect ratio, and can appear on any part of an image. One method that has been used extensively to gather information about objects is multi-scale sliding window (Gonzalez, Villalonga, Ros, Vázquez, & López, 2015). Sliding window is a rectangular region of fixed width and height that slides across an image and the details of each window are passed to an image classifier to look for an object of interest and its position. Although this strategy can try to find out all positions of the objects, it has some shortcoming. Since there are large number of candidate windows, it is computationally expensive and produces too many windows which are redundant. However, if a fixed number of windows are used, there is a possibility of the method providing unsatisfactory regions which may lead to non detections.

Feature extraction: For every window generated, visual features have to be extracted to provide semantic and robust representation. Superpoint (DeTone, Malisiewicz, & Rabinovich, 2018), D2-net (Dusmanu et al., 2019), and LF-Net (Ono, Trulls, Fua, & Yi, 2018) are the most commonly used feature extractors because of their accuracies. Despite their success, some challenges like illumination conditions, diversity of appearances and backgrounds have made it difficult to design a robust feature descriptor manually to describe all kinds of objects with very little error.

Classification: A classifier is needed to distinguish a target object from other objects and to have the representations more semantic, hierarchical and informative for visual recognition. Supported Vector Machine (SVM) (Juang & Chen, 2011), AdaBoost (Freund & Schapire, 2015) and Deformable Part-based Model (DPM) (Felzenszwalb, Girshick, McAllester, & Ramanan, 2011) are the most commonly used classifiers. Among these classifiers, the most flexible model is DPM because it combines object parts with defor-

mation cost to handle severe shape deformations. In DPM, with the help of graphical models, carefully designed low-level features and kinematically inspired part decompositions are joined. Discriminative learning of graphical models also allows high-precision part-based models to be developed for a variety of classes.

The steps stated above have been used to develop shallow architectures and have performed well in PASCAL Visual Object Classes 2005 (VOC) competition (Everingham, Van Gool, Williams, Winn, & Zisserman, 2011) by averaging 0.76mAP, 0.73mAP, and 0.75mAP for SVM, Adaboost, and DPM respectively. Despite these successes, not much gains were being made with respect to accuracies and speeds until Deep Neural Networks (DNNs)(Krizhevsky et al., 2012) were developed. Significant gains were later obtained with the introduction of Regions with Convolutional Neural Network features (R-CNN) (Girshick, Donahue, Darrell, & Malik, 2014). These CNNs have deeper architecture (networks with more than 1 hidden layer) with the ability to learn more complex features than the shallow architectures. Another advantage of these networks is the robust training algorithms used. They allow the networks to learn informative object features without the need to design features manually (LeCun, Bengio, & Hinton, 2015).

Since the development of R-CNN, more advanced models have been suggested which include Fast R-CNN that optimizes classification and bounding box regression tasks together (Girshick, 2015), Faster R-CNN that has an additional sub-network to generate region proposals (S. Ren, He, Girshick, & Sun, 2015), and Single Shot Multi-box Detection that achieves object detection using a fixed-grid regression (Liu et al., 2016). These advancements in detection have yielded improvement in performance as compared to the primary R-CNN and make real-time and accurate object detection feasible (Girshick et al., 2014).

2.2 Overview of Convolutional Neural Networks

Convolutional neural networks (CNNs) are made up of layers of neurons that have learnable weights and biases. Each neuron receives some inputs, calculates a dot product, and optionally follows it with a nonlinear function. This process is repeated layer by layer until the output layer, where the network's prediction is generated. These networks are specifically designed to work with grid-structured inputs that have strong spatial dependencies in local regions of the grid. The most obvious example is the natural image. Natural im-

ages exhibit spatial dependencies where adjacent pixels often have similar colour values. Therefore, the features within an image have dependencies among one another based on spatial distances (Pinaya, Vieira, Garcia-Dias, & Mechelli, 2020).

2.2.1 How Convolutional Neural Networks Work

To understand convolutional neural networks, it is important to first understand how some layers function in these neural network.

2.2.1.1 Convolution Layer

This layer is usually the first layer in a CNN and it can be repeated within the network but with different parameters. Starting with a 7x7 matrix as shown in Figure 2.1, a 3x3 filter/neuron/kernel is introduced and it scans the matrix horizontally from top left to the last window at bottom right (sliding window) in steps of 1 pixel(single stride); the area it scans is called the receptive field as shown in Figure 2.1 (Patin, 2013). The kernel

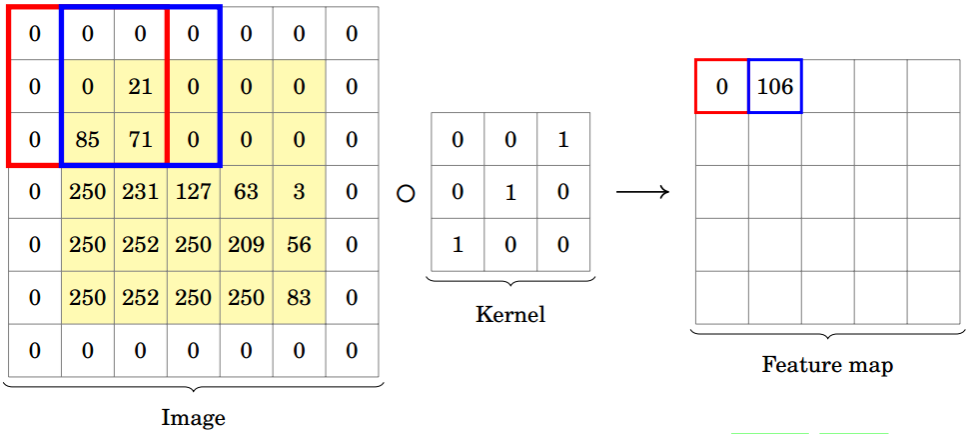


Figure 2.1: How matrix convolution is achieved (Patin, 2013)

used here is a diagonal filter and it will extract any diagonal feature from the image. The values in the first receptive area (first 3x3 matrix from the top-left corner) are multiplied to the value of the kernel, summed up and the answer stored in the first segment of the feature map matrix (4x4). The kernel moves one stride to the right and the process is repeated again as before and the values stored on the second column of the feature map. This process is repeated until the whole input is completely scanned. The values of the feature map will tell if a diagonal line is present or not by simply looking at the numbers; big positive numbers affirm presence while negative or zero number mean absence of

the same. To extract more features like: vertical lines, diagonal lines etc, more filters are introduced and they will all scan the image to generate feature maps which can be analyzed to determine presence or absence of the desired features (Patin, 2013).

2.2.1.2 Activation Layer

An activation layer is usually applied after a convolutional layer to solve the problem of vanishing gradient which slows down the network training process (X. Wang, Qin, Wang, Xiang, & Chen, 2019). The layer also introduces non-linearity to a network that has been computing linear operations in the convolutional layer. The three common functions used include: tanh, sigmoid, and rectified linear units (ReLU): ReLU that has become popular because it helps the network train faster with minimum loss of accuracy (Xu, Wang, Chen, & Li, 2015). This function converts all negative activations to zero using the function shown in equation 2.1.

$$f(x) = \max(0, x) \tag{2.1}$$

where x is the input of the network's neuron. ReLU(Patin, 2013) can be graphically represented as shown in Figure 2.2 where any input value x below zero is outputted as zero but greater than zero is outputted proportionally.

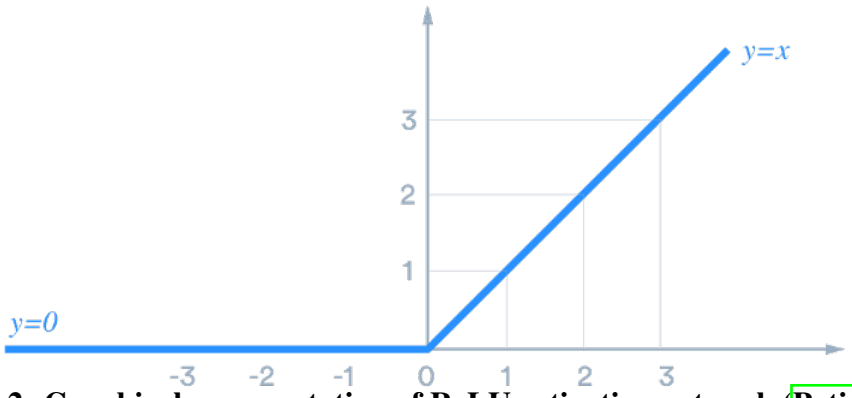


Figure 2.2: Graphical representation of ReLU activation network (Patin, 2013).

2.2.1.3 Pooling Layer

This layer is also called a down-sampling layer. As the name suggests, it helps reduce the amount of information being fed into it but in a way that the most important bits are maintained and it is normally used after an activation layer. The two main reasons for this approach are

- To reduce the cost of computing by reducing the number of parameters the network needs.
- To reduce the chances of over-fitting. This is a phenomenon where the network is exceptional at learning the training data but fails miserably at evaluating the test data or any other data given to it.

The types of pooling layer include; maxpooling, L1-norm, L2-norm and average pooling. Maxpooling is popular and it is achieved by introducing a filter say 2x2 with a stride of the same dimension (two) and it picks the maximum number in the category as it convolves the input as shown in Figure 2.3 (Oquab, Bottou, Laptev, & Sivic, 2014).

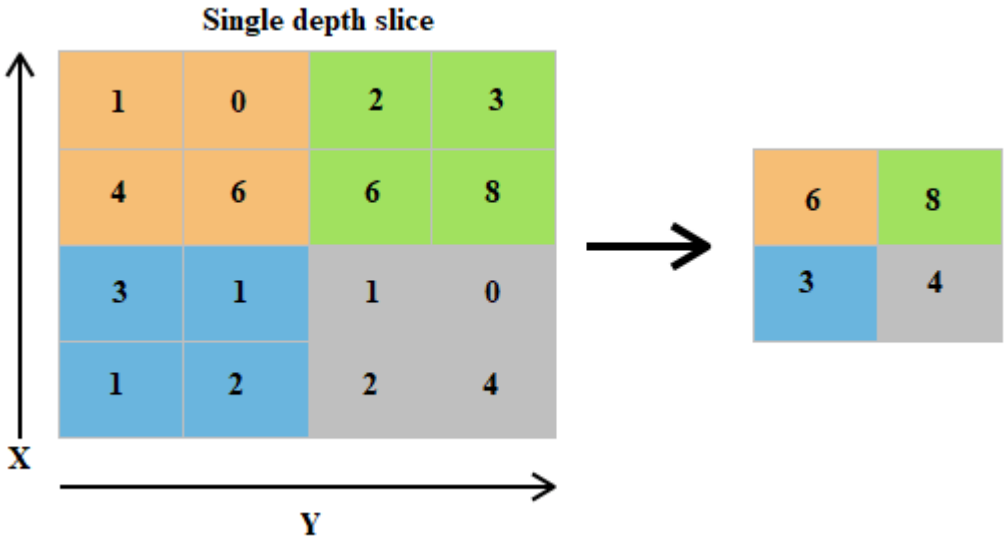


Figure 2.3: How max pooling is achieved.

2.2.1.4 Dropout Layer

This layer helps the network with the over-fitting problem discussed previously. It randomly drops out some of the neurons and their connections inward and outward as shown in Figure 2.4. This technique was proposed by Smirnov et al (Smirnov, Timoshenko, & Andrianov, 2014) and it reduced overfitting but at the expense of training time. To understand the operation of CNN, LeNet-5 architecture depicted in Figure 2.5(Wei, Li, Zhao, & He, 2019) is used. The network had 8 layers but 7 layers contain trainable weights. A

20x20 pixel image was first padded into a 32x32 pixel image. Padding is adding a outer boundary layer that does not affect the original image as shown in Figure 2.6. Padding helps to preserve as much information about the original image so that the network can extract low-level features such as stroke end-points or corners by centering it. The image was then normalized (dividing each element of a matrix by its determinant) so that the white background corresponded to -0.1 and the black foreground to correspond to 1.175. This leads to a mean input of 0 and a variance of about 1 which has been shown to accelerate the learning rate of the network.

The first step in training a neural network is initialising all filters and weights with random values. The input image (for instance a picture of an apple) is then fed into the network and it goes through convolution, ReLU and pooling operations (forward propagation) after which it outputs the probabilities of the image being a member of the classes, for example [boat=0.4, cat=0.3, cow=0.1, apple=0.2]. The outputs are usually random the first time since weights are initialized randomly. A error is then calculated using Equation. 2.2.

$$E = \sum \frac{1}{2}(TP - OP)^2 \tag{2.2}$$

where E is error, TP is target probability, and OP is output probability.

Back propagation (Cilimkovic, 2015) is then used to calculate the gradient of the error with respect to all weights in the network and gradient descent (Bottou, 2010) is used to update all filter weights to minimize the output error. The weights are then adjusted

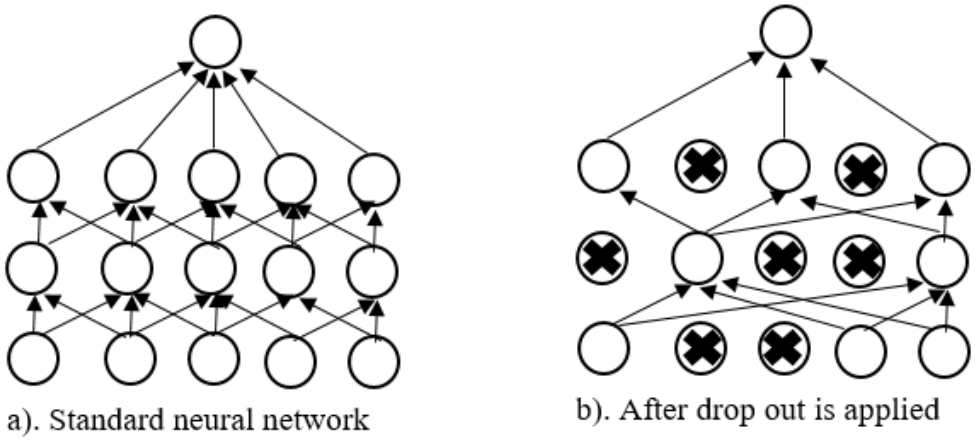


Figure 2.4: Drop-out pictorial representation (Smirnov et al., 2014).

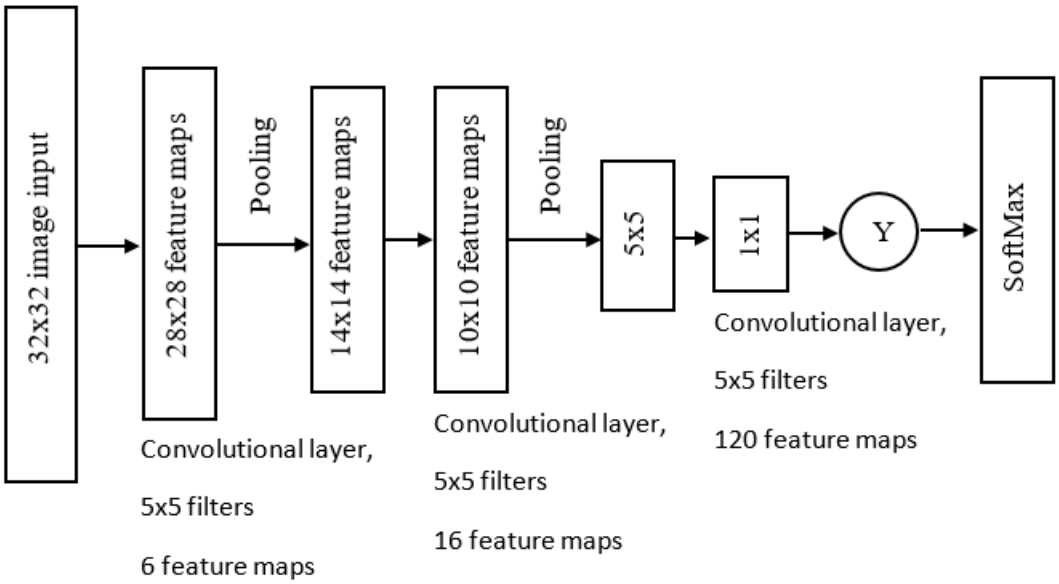


Figure 2.5: Architecture of LeNet-5, a convolutional neural network for digital recognition(Wei et al., 2019).

35	19	25	6
13	22	16	53
4	3	7	10
9	8	1	3

(a) A 4x4 matrix before zero padding.

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

(b) A zero padded matrix.

Figure 2.6: Zero padding example.

in proportion to their contribution to the total error which would now give an output of [0.1,0.1,0.1,0.7] which is closer to [0,0,0,1] meaning the network has learnt to classify an apple in an image. This whole process is repeated with all images and the more the images the better trained a network gets. A summary of the working of these layers is shown in Figure 2.7

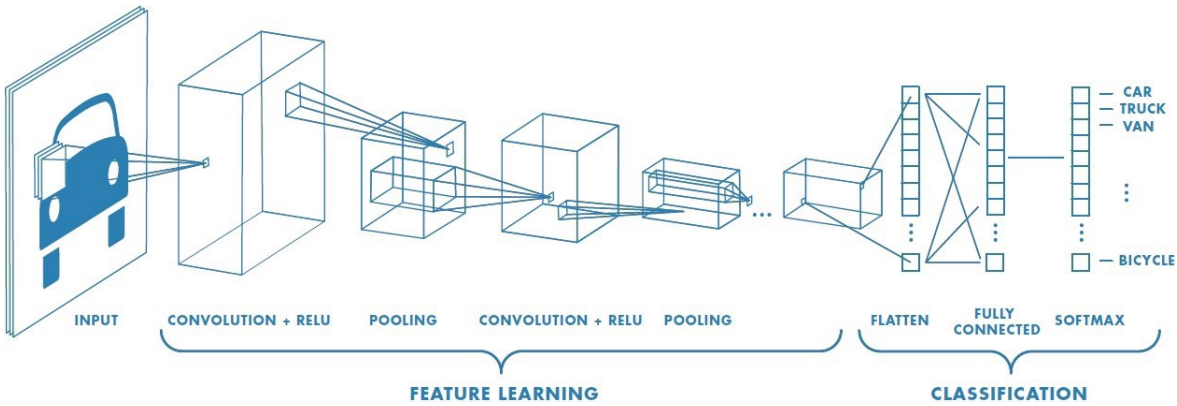


Figure 2.7: Example of a typical CNN architecture showing the key components of the network.

2.3 Generic Object Detection

In generic object detection, the aim is to locate and classify existing objects in an image and label them with a bounding box to show the confidences of existence in percentage form. Generic object detection methods can be broadly grouped into two types, region proposal or regression based. In region proposal, the traditional object detection pipeline is used, that is generating region proposals and then classifying each proposal into different object classes. Regression based detection involves viewing object detection as a regression or classification problem, adopting a unified framework to achieve categories and locations. Region proposal based methods include R-CNN (Girshick et al., 2014), Spatial pooling pyramid network (SPP-net) (He, Zhang, Ren, & Sun, 2015), Fast R-CNN (Girshick, 2015) and Faster R-CNN (S. Ren et al., 2015). The regression based methods mainly includes You Only Look Once (YOLO) (Redmon, Divvala, Girshick, & Farhadi, 2016) and SSD (Liu et al., 2016).

2.4 Region Proposal

This is a two-step process, mimicking the human brain's attention mechanism to some point, which scans of the whole scene first then focuses on the regions of interest. Some related works include (Hinton, Krizhevsky, & Wang, 2011; Sermanet et al., 2013; Taylor, Spiro, Bregler, & Fergus, 2011) the most representative being Overfeat network (Sermanet et al., 2013). This network merges CNN and sliding window method which predicts the bounding boxes directly from locations of the top most feature map after obtaining the

confidences of underlying object categories. The networks discussed in the proceeding section employ this framework but with different iterations.

2.4.1 R-CNN

This network was developed to improve quality of candidate bounding boxes and to extract high-level features [15]. The R-CNN was able to achieve a mean average precision (mAP) of 53% which was an improvement over DPM's best result of 42% (X. Ren & Ramanan, 2013) on PASCAL VOC 2007. Figure 2.8 shows the three main stages of an R-CNN detection process.

- **Region proposal generation**. Selective search (Uijlings, Van De Sande, Gevers, & Smeulders, 2013) is used to generate almost 2000 region proposals for every image. The selective search method depends on simple bottom-up grouping and saliency cues to generate more accurate arbitrary sized candidate boxes quickly and area to search for objects (Deng et al., 2010; Felzenszwalb et al., 2011).

- **CNN feature extraction**. Every region proposal is warped into a fixed resolution and a CNN module utilized to extract a 4096 pixel dimensional feature. Due to large learning capacity, dominant expressive power and hierarchical structure of CNNs, a high-level, semantic and robust feature representation for each region proposal is obtained.

- **Classification and localization**. Support Vector Machines (SVM) is used to score different regions proposed (Juang & Chen, 2011). The regions are assigned positive scores but the background is given negative scores. The scored regions are then adjusted using bounding box regression and a greedy non-maximum suppression

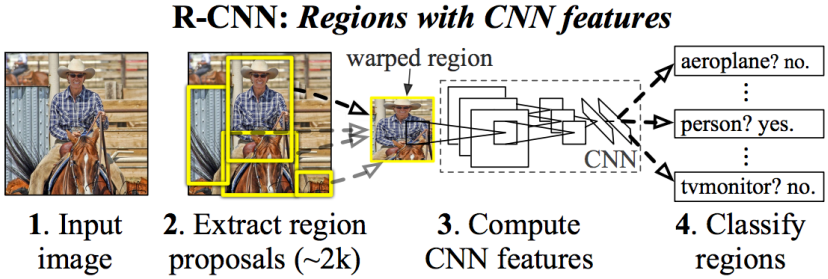


Figure 2.8: The flowchart of R-CNN (Girshick et al., 2014).

(NMS) is used as a filter to produce final bounding boxes (Rothe, Guillaumin, & Van Gool, 2014).

The R-CNN also allows for pre-training the network when there is insufficient labelled data. This is achieved by first conducting supervised pre-training on ILSVRC dataset followed by application-specific fine-tuning (Girshick, 2015).

Despite improved accuracies and speed as compared to previous methods, some challenges are still exhibited:

- The presence of a fully connected (FC) layer requires the input image of fixed size which directly leads to re-computation of the whole CNN for each evaluated region, taking a lot of time.

- The whole process is multi-stage requiring fine tuning of the convolutional network for object proposal, the softmax classifier has to be replaced by SVM to fit the features and the bounding box regressors are also trained. These stages makes the network slow.

- The speed being low makes the training time expensive computationally since a lot of features are generated and have to be stored in memory discs.

- Although Selective Search has been proven to generate region proposals with high recall, the region proposals obtained are mostly redundant meaning a lot of time is wasted. The number of layers and stride also increased in Selective Search which led to loss of resolution and hence making localization a challenge.

To solve these challenges, techniques such as multi-scale combinatorial grouping (Arbeláez, Pont-Tuset, Barron, Marques, & Malik, 2014) searches different scales of the image for multiple hierarchical segmentations and combinatorially groups different regions to produce proposals. This technique improved the time taken to develop proposals from 20 s per image to 17 s per image. Instead of extracting visually distinct segments, the edge boxes method (Zitnick & Dollár, 2014) adopts the idea that objects are more likely to exist in bounding boxes with fewer contours straggling their boundaries. A recall of 96% was obtained at 0.5 overlap and 96% at 0.7 overlap which demonstrated better recall rate than R-CNN. Also some studies such as DeepBox (Kuo, Hariharan, & Malik, 2015),

and SharpMask (Pinheiro, Lin, Collobert, & Dollár, 2016) tried to refine pre-extracted region proposals to remove unnecessary ones and obtained only valuable ones. DeepBox increased precision of R-CNN by 4 mAP while attaining speeds of 3.8 fps which is far from real time speeds (30 fps). SharpMask was 50% faster than DeepBox further improving speed of R-CNN with an increase in recall rate precision by 10%-20%. The speed was still not fast enough to attain real time speeds.

Attempts to solve Inaccurate localisation have also been made. Bayesian optimization-based search algorithm has been used to guide the regressions of different bounding boxes sequentially, and trained class specific CNN classifiers with a structured loss to penalize the localization inaccuracy (Zhang, Sohn, Villegas, Pan, & Lee, 2015). Object detection for red-green-blue-distance (RGB-D) images was improved with semantically rich image and depth features (Gupta, Girshick, Arbeláez, & Malik, 2014), and learned a new geocentric embedding for depth images to encode each pixel. The combination of object detectors and super-pixel classification framework poses a promising result on semantic scene segmentation task.

2.4.2 SPP-net

Fully connected (FC) layers take a fixed-sized input and it prompted R-CNN to crop each region proposal to the same size. This meant a loss of information since some objects might exist in the cropped area and also cropping might introduce some errors to the output feature. This cropping lead to inaccuracies in the final result (Lee, Kim, & Oh, 2016).

To solve this problem, He et al. (He et al., 2015) took the theory of spatial pyramid matching (SPM) (Lazebnik, Schmid, & Ponce, 2010; Perronnin, Sánchez, & Mensink, 2010)into consideration and proposed a CNN architecture named SPP-net (He et al., 2015). The SPM takes several finer to coarser scales to partition the image into a number of divisions and aggregates quantized local features into mid-level representations as shown in Figure 2.9. In the Figure, the input image is scanned and features of different pooling levels are generated that is L0 (1×1 grid), L1 (2×2 grid) and L2 (4×4 grid). These features contain object information that is used by the fully connected layers (FC layers) to classify the objects in the image.

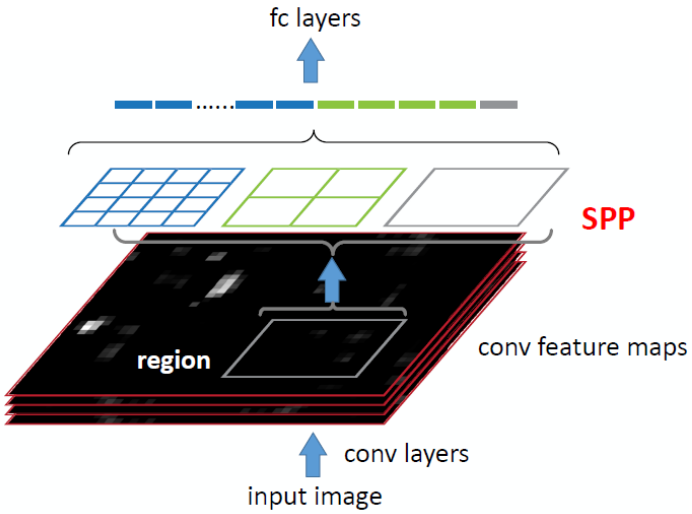


Figure 2.9: SPP-net architecture (He et al., 2015)

Different from R-CNN, SPP-net reuses feature maps of the 5-th convolutional layer (conv5) to project region proposals of arbitrary sizes to fixed-length feature vectors. The feasibility of the reusability of these feature maps is due to the fact that the feature maps not only involve the strength of the local response, but also have relationships with their spatial positions (He et al., 2015). The layer after the final convolutional layer is referred to as spatial pyramid pooling layer (SPP layer). If the number of feature maps in conv5 is 100, taking a 3-level pyramid, the final feature vector for each region proposal obtained after SPP layer has a dimension of $100\times(1^2 + 2^2 + 4^2) = 2100$.

The SPP-net not only produces better results with correct estimation of different region proposals irrespective of scale, but also improves detection efficiency in testing period with the sharing of computation cost before SPP layer among different proposals.

2.4.3 Fast R-CNN

Although SPP-net has achieved impressive improvements in both accuracy and efficiency over R-CNN, it still has some notable drawbacks. The SPP-net takes almost the same multi-stage pipeline as R-CNN, including feature extraction, network fine-tuning, SVM training and bounding box regressor fitting. So an additional expense on storage space is still required. Additionally, the convolutional layers preceding the SPP layer cannot be updated with the fine-tuning algorithm introduced in (He et al., 2015). As a result, an accuracy drop-off is realized. To solve this limitation with SPP, a multi-task loss function

on classification and bounding box regression was introduced and a CNN architecture named Fast R-CNN was developed (Girshick, 2015), as depicted in Figure 2.10.

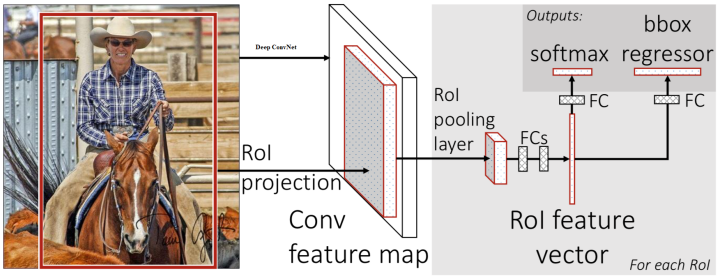


Figure 2.10: Fast RCNN architecture for object detection (Girshick, 2015)

In the image, the input image is fed to the Deep ConvNet for image classification and the RoI (region of interest) projection for region of interest generation. Here the output is a feature map that is sent to RoI pooling for compression and FCs (fully connected layer) for weight learning. The output of the FCs is a RoI feature vector which is a 7×7 grid which is fed to two FCs, one for classification and another one for bounding box regressor.

Similar to SPP-net, the whole image is processed with convolutional layers to produce feature maps. Then, a fixed-length feature vector is extracted from each region proposal with a region of interest (RoI) pooling layer. The RoI pooling layer is a special case of the SPP layer, which has only one pyramid level. Each feature vector is then fed into a sequence of FC layers before finally branching into two output layers. One output layer is responsible for producing softmax probabilities for all object classes and the background categories while the other output layer generates bounding box positions with four real-valued numbers.

To improve on accuracy, L1 loss function is adopted to fit bounding-box regressors. To accelerate the pipeline of Fast R-CNN, computation and memory are shared by RoIs from the same image in the forward and backward pass. Since much time is spent in computing the FC layers during the forward pass (Girshick, 2015), Singular Value Decomposition (SVD) (Xue, Li, & Gong, 2013) is used to compress large FC layers and to accelerate the testing procedure.

In Fast R-CNN, regardless of region proposal generation, the training of all network layers can be processed in a single-stage with a multi-task loss function. Multi-task loss function

optimizes both classification and regression events at the same time unlike loss function like Softmax that can only optimize either of these events one at a time. This saves on storage space, and improves both accuracy and efficiency during training.

2.4.4 Faster R-CNN

Despite trying to generate candidate boxes with biased sampling, object detection networks rely on additional methods, such as selective search and Edgebox, to generate a candidate pool of isolated region proposals. Region proposal computation is also a bottleneck in improving efficiency. To solve this problem, Ren et al (S. Ren et al., 2015) introduced an additional Region Proposal Network (RPN), which acts in a nearly cost-free way by sharing full-image convolutional features with detection network.

The RPN is achieved with a fully-convolutional network, which has the ability to predict object bounds and scores at each position simultaneously. Similar to Selective Search (Uijlings et al., 2013), RPN takes an image of arbitrary size to generate a set of rectangular object proposals. The RPN operates on a specific convolutional layer with the preceding layers shared with object detection network.

The architecture of RPN is shown in Figure 2.11. The network slides over the conv feature map and fully connects to an n×n spatial window. A low dimensional vector (512-d for VGG16) is obtained in each sliding window and fed into two sibling FC layers, namely box-classification layer (cls) and box-regression layer (reg). This architecture is implemented with an n×n convolutional layer followed by two 1×1 convolutional layers. To increase non-linearity, ReLU is applied to the output of the n×n convolutional layer (Li & Yuan, 2017).

The regressions towards true bounding boxes are achieved by comparing proposals relative to reference boxes (anchors). In the Faster R-CNN, anchors of 3 scales and 3 aspect ratios are adopted. The loss function used is given by Equation. 2.3(S. Ren et al., 2015).

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*), \tag{2.3}$$

where p_i represents the predicted probability of the *i*-th anchor being an object. Ground truth p_i^* is 1 if the anchor positive, else, 0. t_i stores the four coordinates of the predicted

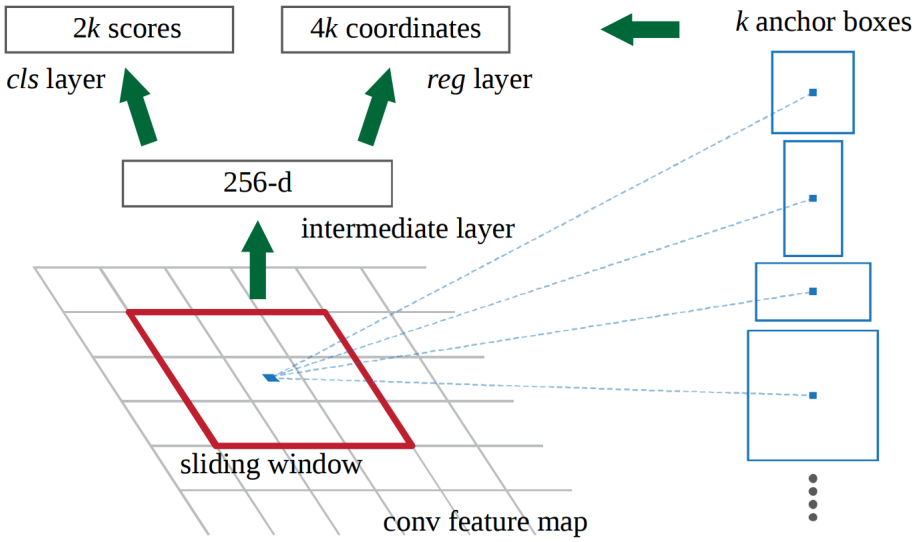


Figure 2.11: RPN in Faster RCNN (S. Ren et al., 2015)

bounding box. t_i^* represents the ground truth box. L_{cls} is a binary loss function. L_{reg} is a smooth L_1 loss function similar to one used in Fast R-CNN. L_{cls} and L_{reg} are normalized by number of anchors locations N_{reg} and mini-batch size N_{cls}.

With the proposal of Faster R-CNN, region proposal based CNN architectures for object detection can really be trained in an end-to-end way. A frame rate of 5 FPS (Frame Per Second) on a GPU is achieved with state-of-the-art object detection accuracy which is an improvement as compared to Fast R-CNN whic has 0.4 fps on PASCAL VOC 2007 and 2012 (Li & Yuan, 2017).

2.5 Regression Based Framework

In region proposal network, there are several stages including region proposal generation stage, feature extraction, classification and bounding box regression. These many stages hinder these networks from attaining real time detection. Regression based networks use one-step frameworks from input to output hence reducing time spent and consequently attaining real time detection (J. Huang et al., 2017).

2.5.1 You Only Look Once (YOLO)

Redmon et al. (Redmon et al., 2016) proposed a framework which makes use of the whole topmost feature map to predict both confidences for multiple categories and bounding boxes. As shown in Figure 2.12, YOLO divides the input image into an $S \times S$ grid and each grid cell is responsible for predicting the object centred in that grid cell. Each grid cell predicts B bounding boxes and their corresponding confidence scores. Formally, confidence scores are defined as $Pr(Object)\text{x}IOU_{pred}^{truth}$, which indicates how likely there exist objects $Pr(Object) \geq 0$ and shows confidences of its prediction IOU_{pred}^{truth}. Where IOU stands for intersection over union. At the same time, regardless of the number of boxes, C conditional class probabilities $Pr(Class_i|Object)$ should also be predicted in each grid cell. It should be noted that only the contribution from the grid cell containing an object is calculated.

At test time, class-specific confidence scores for each box are achieved by multiplying the individual box confidence predictions with the conditional class probabilities as shown in

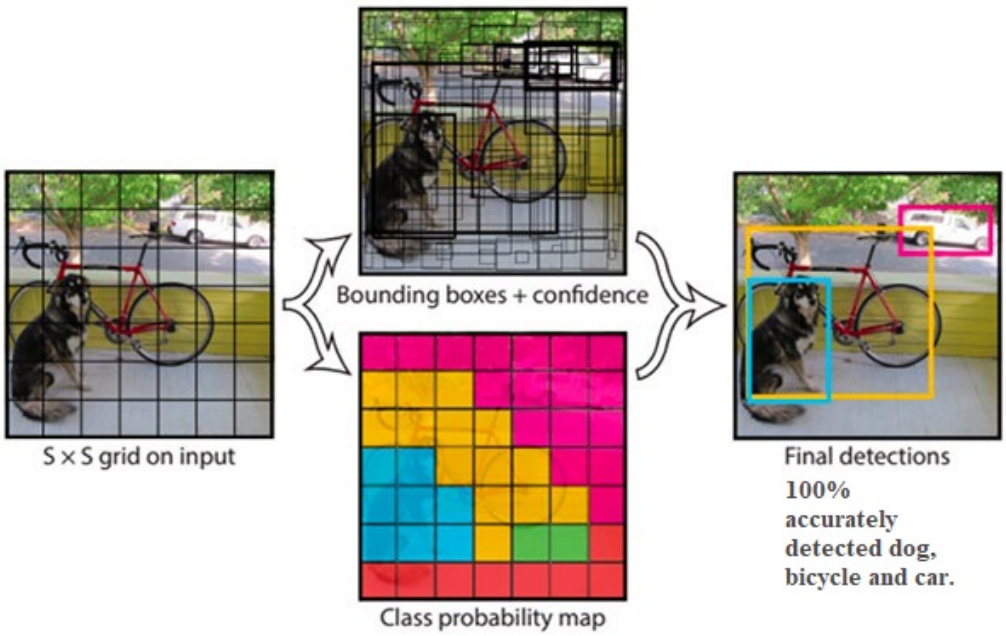


Figure 2.12: Illustration of YOLO (Redmon et al., 2016).

Equation 2.4(Redmon et al., 2016).

$$\begin{aligned} &Pr(Object) \times IOU_{pred}^{truth} \times Pr(Class_i|Object) \\ &= Pr(Class_i) \times IOU_{pred}^{truth} \end{aligned} \tag{2.4}$$

During training, the loss function in Equation 2.5(Redmon et al., 2016) is optimized,

$$\begin{aligned} L = \lambda_{coord} &\sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{obj} [(x_i - \hat{x}_i)^2 + ((y_i - \hat{y}_i)^2)] + \\ \lambda_{coord} &\sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{obj} [(w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2] + \\ &\sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{obj} (C_i - \hat{C}_i)^2 + \\ \lambda_{noobj} &\sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{noobj} (C_i - \hat{C}_i)^2 + \\ &\sum_{i=0}^{S^2} O_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \tag{2.5}$$

Where cell *i*, (x_i, y_i) denotes the centre of the box relative to the bounds of the grid cell, (w_i, h_i) are the normalized width and height relative to the image size, C_i represents confidence scores, O_i^{obj} indicates the existence of objects and O_{ij}^{obj} denotes that the prediction is conducted by the j^{th} bounding box predictor (Redmon & Farhadi, 2017). The loss function only penalizes classification errors when an object is present in that grid.

The YOLO network consists of 24 convolutional layers and 2 FC layers, of which some conv layers construct ensembles of inception modules with 1×1 reduction layers followed by 3×3 convolutional layers. The network can process images in real-time at 45 FPS and a simplified version Fast YOLO can reach 155 FPS with better results than other real-time detectors. Furthermore, YOLO produces fewer false positives on background, which makes the integration with Fast R-CNN possible.

2.5.2 Single Shot Multibox Detection (SSD)

You only Look Once (YOLO) has a difficulty in dealing with small objects in groups, which is caused by strong spatial constraints imposed on bounding box predictions (Redmon et al., 2016). Meanwhile, YOLO struggles to generalize objects in new/unusual aspect ratios/ configurations and produces relatively coarse features due to multiple downsampling operations.

To solve these challenges, Liu et al.(Liu et al., 2016) proposed a Single Shot MultiBox Detector (SSD), which was inspired by the anchors adopted in MultiBox (Erhan, Szegedy, Toshev, & Anguelov, 2014), RPN (S. Ren et al., 2015) and multi-scale representation (Bell, Lawrence Zitnick, Bala, & Girshick, 2016). Given a specific feature map, instead of fixed grids adopted in YOLO, the SSD takes advantage of a set of default anchor boxes with different aspect ratios and scales to discretize the output space of the bounding boxes. To handle objects with various sizes, the network fuses predictions from multiple feature maps with different resolutions.

The architecture of SSD is demonstrated in Figure 2.13. Given the VGG16 backbone architecture, SSD adds several feature layers to the end of the network, which are responsible for predicting the offsets to default boxes with different scales and aspect ratios and their associated confidences. The network is trained with a weighted sum of localization loss (e.g. Smooth L1) and confidence loss (e.g. Softmax). Final detection results are obtained by conducting NMS on multi-scale refined bounding boxes. Integrating with hard negative mining, data augmentation and a larger number of carefully chosen default anchors, SSD significantly outperforms the Faster R-CNN in terms of accuracy on PASCAL VOC and COCO, while being three times faster. The SSD300 (input image size is 300×300) runs at 59 FPS, which is more accurate and efficient than YOLO. However, SSD performs poorly when dealing with small objects(objects with pixel area less than 1024 pixels), which can be improved by adopting better feature extractor backbone (e.g. ResNet-50), adding deconvolution layers with skip connections to introduce additional large-scale context (Fu, Liu, Ranga, Tyagi, & Berg, 2017) and designing better network structure (e.g. Stem Block and Dense Block) (Shen et al., 2017).

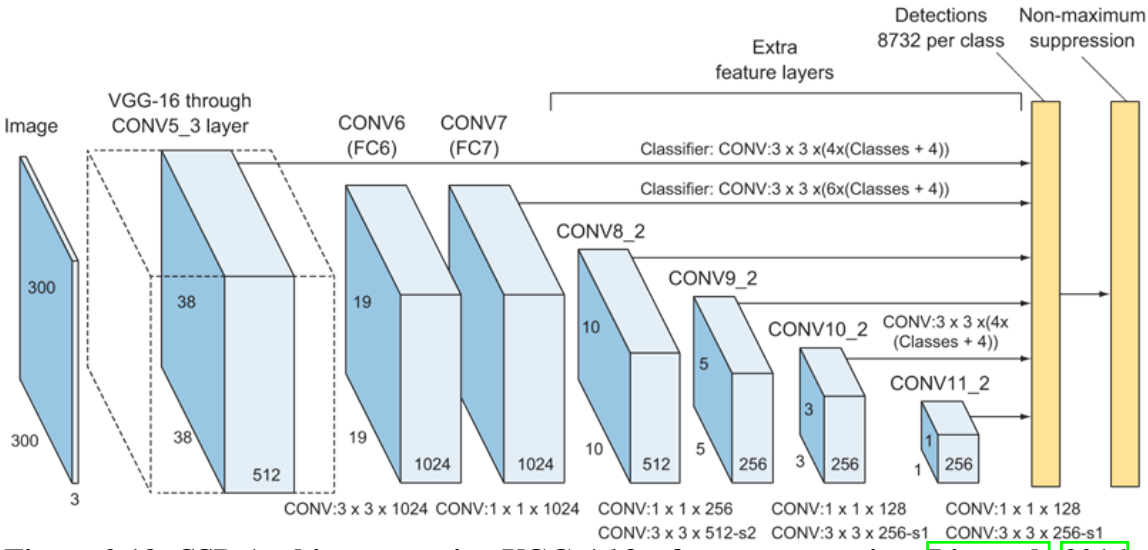


Figure 2.13: SSD Architecture using VGG-16 for feature extraction (Liu et al., 2016).

2.6 Google Cloud Machine Learning Engine

This is a dedicated platform offered by Google for machine learning and offers a way to link Tensor Processing Units (TPUs) with the model being trained (Ciaburro, Ayyadevara, & Perrier, 2018). The platform's servers is built from Ubuntu 16.0 operating system with Python 3.6 as the default programming language.

Tensorflow(Abadi et al., 2016) is a major machine learning library that is used by researchers and developers in machine learning. It offers an object detection Application Programming Interface, API (a set of subroutine definitions, communication protocols, and tools for building software which allows for a developer to come up with any object detection model) (J. Huang et al., 2017). The images are first collected and labelled and uploaded to the Google Cloud Machine Learning Engine (GCMLE) setup for training.

A decision on either training a network from scratch or using a pre-trained network that has learnt basic objects and customizing it to fit desired application has to be made at this point. Starting from scratch is advantageous because it is flexibility in design although it takes a lot of time and needs a lot of computing power. The other option needed one to get a project almost similar to the one undertaken and either add or remove details like the number of layer or neurons.

2.7 Related Work

Researchers have been trying to model complex systems using techniques like system identification and function approximation methods. Complex problems like controlling a helicopter to fly in acrobatic manoeuvres is one of the challenges where system identification has been utilised but without much success.

Punjani et al. (Punjani & Abbeel, 2015) however used a CNN with functional approximating algorithm and rectifiers to model a radio-controlled helicopter. An expert flew the helicopter through some manoeuvres ranging from basic to complex aerobatic ones and these moves were used as training data. The data included the keys pressed by the operator and the corresponding velocity and the direction achieved by the helicopter. The data was then fed to the CNN which came up with a relationship between the keys pressed and the desired output. The network was then compared to three state-of-the-art methods and it out performed them by about 60% in obtaining helicopter dynamics. Despite the impressive performance, the CNN was not efficient in training since the researchers needed a 6 core Intel i7 server with 32GB RAM yet the network size is smaller than plain region proposal networks (Dai, Li, He, & Sun, 2016; S. Ren et al., 2015). The system was also not able to achieve autonomous operation mainly because it could not achieve real time performance.

Neverova (Neverova, Wolf, Taylor, & Nebout, 2014) used a Faster R-CNN to model how long between a driver's head movement and a manoeuvre occurring at varied vehicle speed. Upon testing, the network was able to make predictions every 0.8s based on the preceding 5 s of data and anticipated manoeuvres about 3.5 s before they occurred, with 90.5 % accuracy. This research was able to achieve 24 fPS which is a pointer on how to achieve real time speeds (30 fps) while still achieving high detection accuracy. This was a guide to this research on methods of collecting training data and handling noise introduced to the vision sensor. Significant number of researchers have used function approximating models in different domains and they have been grouped by (Pierson & Gashler, 2017) into (a) detection and perception, and (b) grasping objects and manipulation.

In detection and perception, convolutional neural networks (CNN) have been favoured by researchers because the input images do not need to be prepared before hand which is cheaper and consumes less time. Such preparation include developing feature vectors for

input images that have to be done manually and are time consuming. These feature vectors may also need experts to analyse them hence increasing overall cost of specialised labour (LeCun et al., 2015).

Mariolis et al. (Mariolis, Peleka, Kargakos, & Malassiotis, 2015) developed a system that uses SPP-net to estimate pose and recognise deformed garments hanged from a single point autonomously using a robotic arm and a depth vision sensor. The system was trained using pants, towels and shirts of different shapes, sizes, and material properties. The system was able to achieve 100 % accuracy in recognition and was further able to predict grasping point for these article of clothing with an error of 5 cm. Comparing these results to support vector machine (SVM) implementation of the same task, the CNN outperformed SVM by 2 % showing its superiority, though by a narrow margin. The system was not able to achieve real time speeds and the picking and placing feature was not incorporated. The network also only accepted input images of 160x64 pixels which meant a sub-routine had to be used to down case images from a camera. This further hindered real time performance of the network. Partial occlusion was discussed as a challenge when picking garments especially when the robot was tasked to identify and pick multiple garments but a solution was not proposed. The researchers did not look into the impact of change in lighting conditions or change in camera angle.

Having achieved high degree of accuracy with object recognition, other researchers looked at the ability of vision based pick and place robots to handle tasks it had never seen before. Yang et al. (Yang, Li, Fermuller, & Aloimonos, 2015) trained a Faster R-CNN to identify common kitchen items and generate grasping points for these forty eight (48) items after watching eighty eight (88) YouTube videos. These videos were new to the robot and were not made with the robot in mind. The system achieved 79 % object recognition and 91 % grasp classification accuracy. Neural networks have also been used for path planning using vision sensors like in research by Chen et al. (W. Chen et al., 2014) which used a RCNN to identify existence of a door and its pose. To identify pose of the doors, the researchers looked into the effect of change of camera angle and it was observed that precision decreased with increase in angle from 0°. This was because the items used in the experiment were not symmetrical and as the angle increased the network could no longer distinguish the object precisely. Precision dropped by 0.4 mAP from 0° to 90°. The performance of the network with respect to real time speed was not evaluated supposedly

because speed was not a strength of Faster R-CNN.

On grasping and object manipulation, CNN was used to categorize five three-dimensional objects laying on a surface and identify their orientation. The objects were known before hand and the task was limited to object recognition and pose estimation only leaving grasp planning to simply positioning a parallel gripper at the respective object's centre. The impact of varying light intensity on grasping was also examined. The system was able to achieve object recognition and grasping accuracy of over 90 % with a light source of 1300 lm. The lowest grasping accuracy was at 60 % when a 500 lm light source was shone on the objects. It was discussed that increasing the light intensity had a direct relationship with increase in grasping accuracy. Faster R-CNN was used by Lenz et al (Lenz, Lee, & Saxena, 2015) to develop grasping points for objects new to the network by using a red green blue and distance (RGB-D) camera. The network generated numerous potential grasping points then the best was chosen. This system was tested on PR2 and Baxter robots and they achieved accuracies of 89 % and 84 % respectively as compared to state-of-the art algorithm that had 31 %. The challenge of attaining real time speeds and inability to work effectively in conditions of partial occlusion was described and the authors proposed using a regression based detection to solve these challenges.

Levine et al. (Levine, Pastor, Krizhevsky, Ibarz, & Quillen, 2018) trained an Fast-RCNN to evaluate the potential of a particular robot motion for successfully grasping common office objects from image data, and used a second network to provide continuous feedback through the grasping process. Inspired by hand-eye coordination in humans, the system was robust to object movement and uncertainty in gripper mechanics. To achieve these results, two networks were used which made the whole system commutatively intense and the speeds were close to 20 fps which was short of real time (30 fps). Challenges of partial occlusion were discussed and it was pointed out that a network with a large dimension fully connected layer would increase the networks performance when exposed to the uncertainty.

Huang et al. (P.-C. Huang & Mok, 2018) used YOLO (Redmon & Farhadi, 2017) to develop an autonomous pick and place robot for warehouses and could handle objects never seen before. The system achieved high object detection accuracies and also real time speeds even in conditions of uncertainty like poor lighting (450lm and below) and partial occlusion. Despite these results, the system still had challenges identifying small objects

Table 2.1: Comparison between SSD with Resnet-18 and VGG16

Parameter	SSD with Resnet-18	SSD with VGG16
Light intensity with 450 lm bulb (%)	59	40
Partial Occlusion-50% (%)	71	65
Real time speeds (fps)	37	35

(area of less than 1024 pixels) and medium sized objects (area of between 1024 and 9216 pixels) mainly because of the incapabilities of the YOLO network. The SSD network designed by Liu et al. (Liu et al., 2016) had better performance with small and medium sized objects as compared to YOLO but research by Chen et al. (S. Chen et al., 2019) has shown improvements of up to 5% in detection accuracies by replacing VGG16 with Resnet-50 as the base network in SSD. Uncertainties in terms of light intensity and occlusion were also examined by Xiong et al.(Xiong, Yao, Ma, & Wu, 2019) using SSD with Resnet-18 and Liu et al. (Liu et al., 2016) as well as real time speeds as shown in Table 2.1. From the table, SSD with Resnet-18 showed improved performance than SSD with VGG16 in terms of object detection under different conditions while still achieving better real time speeds. A variation of SSD that used Resnet-50 had not been evaluated with respect to these uncertainities and specifically in robotics though it would be hypothesised that its performance would rival the SSD version that used Resnet-18.

2.8 Research Gap

Convolutional neural network (CNN) are built with a back born network for classification of objects. To gain performance advantages like accuracy and speed, researchers have interchanging these back born networks. In robotics this has also been done and focus has been on improving object detection accuracy and speed. The CNNs that have been modified to improve performance include fast R-CNN, faster R-CNN and even regression based CNNs like Single Shot multi-box detector (SSD). So far, researchers have interchanged the back bone VGG16 network in SSD with networks like Mobinet and Inception.

Advancements in feature extraction have improved the performance of Resnet-50 in terms of speed and accuracy and it has been shown that it out performed networks like VGG16. Speed and object detection in pick and place robotics can be improved by developing a SSD model that uses Resnet-50 as the backbone object classifier.

CHAPTER THREE

EXPERIMENTAL DESIGN AND METHODOLOGY

3.1 Overview

In this chapter, a single-shot multibox detector (SSD) is developed, trained, and tested for real-time object detection. A robotic arm control system is also developed and coupled to the object detection system for pick and place. The whole system was tested under different lighting conditions, viewing angles, and partial occlusion.

3.2 Methodology Flow Chart

Figure 3.1, shows a flow chart of the steps followed to develop the object detection pick and place robotic system. The first step being collection of the images for training the neural network to identify the object followed by setting up the Google cloud for the actual training and evaluation. Once training was done, the network was tested to ensure that it met the requirements for the task such as mAP, real times speed, and bounding box accuracy. Robot vision was the next step and it involved how the robot perceives the objects and how the location information was going to be processed by the inverse kinematics. The inverse kinematics for the replica robot was calculated since some dimensions did not match the original robot; hence, the kinematics in literature could not be used. An inverse

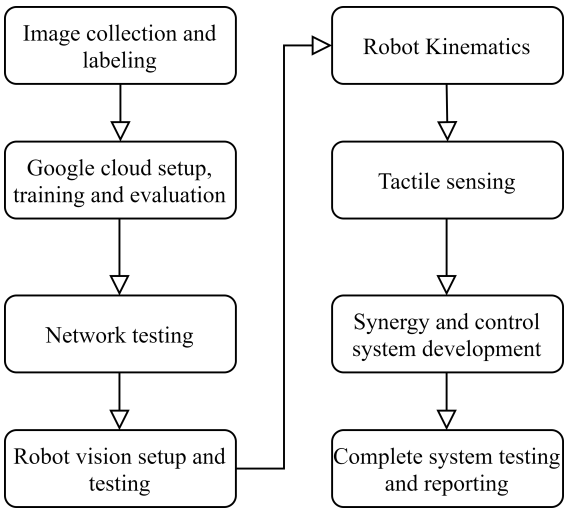


Figure 3.1: A flow chart of the system configuration procedure

kinematic formula was generated and tested for the IRB1400 replica robot. Additionally, the force needed to grasp all objects was finally obtained.

3.3 Image Collection and Preparation

The pick-and-place robot should be versatile and be used with as many objects as possible. As such, thirteen objects from different classes were selected and these objects are depicted in Table 3.1. The choice of the objects was based on the texture (testing gripping

Table 3.1: List of objects used for image collection and object detection.

S/n	Object
1	Toothpaste
2	Toothbrush
3	Onion
4	Orange
5	Insulating tape
6	Phone charger
7	Remote control
8	Match box
9	Flash light
10	Tomato
11	Plastic toy
12	Pen
13	Tamarillo

ability of the robot), shape (testing the ability of object detection algorithm to distinguish different shapes, make accurate detections grasping ability of the robotic arm), and colour (for the object detection). This was so as to extensively evaluate the performance of the whole system. The classes of objects chosen have their packaging automated and it was to show that the over all system was versatile enough for any industry.

The photos of the objects were taken in different lighting conditions (450 lm and 1600 lm), camera positions (0° and 45°), and occlusions (0% and 50%). A total of three hundred and forty seven (347) images were taken. The images were taken with one object per image at first to help the network identify the object, then the number of objects in an image were increased progressively as shown in Figure 3.2. This helped the network to be able to distinguish many objects in one image which improves the network's accuracy. The images were then rotated through angles of 45°, 90°, 180° and 270° to increase the size of the training dataset which in the end was one thousand three hundred and eighty eight

Figure 3.2: A mixture of single and multiple objects in one image that was used for training.

(1388) images. 0° was the angle the camera took the original images. Training on this specific network needed images of 640x640 pixels for smooth training. The camera used (HP 15 mega pixels webcam) to take the 347 images was recording at 4000x3000 pixels and resizing had to be done without compromising the pixel quality and aspect ratio.

The next step was to annotate all of the images using labelImg software (Tzutalin, 2015). Bounding boxes were drawn around the objects in every image and an appropriate label chosen. The labelling had to be consistent so as to avoid confusion of the network during training. The images were then split into training (80%) and testing (20%) as guided by literature in (Dobbin & Simon, 2011).

3.4 SSD Resnet-50 Implementation

As mentioned, the original SSD had a VGG16 backbone but Resnet-50 performed better than it speed and accuracy wise. Figure 3.3 shows a block diagram of the SSD-VGG16 implementation. Conv4_3 represents the VGG16 convolutional layer which needed to be

replaced. The fully connected layer (Conv7) was removed to avoid repetition of this layer in the Resnet50 implementation. The Softmax layer (Conv8_2) which reduced the information size in VGG16 was also removed. To implement Resnet-50, three convolutional layer were added:Conv_3 of dimensions $38\times38\times128$, Conv_4 of dimensions $19\times19\times256$ and Conv_5 of dimensions $10\times10\times512$ as shown in Figure 3.4. These layers represented the Resnet-50 layer, feature fission layer and a fully connected layer respectively.

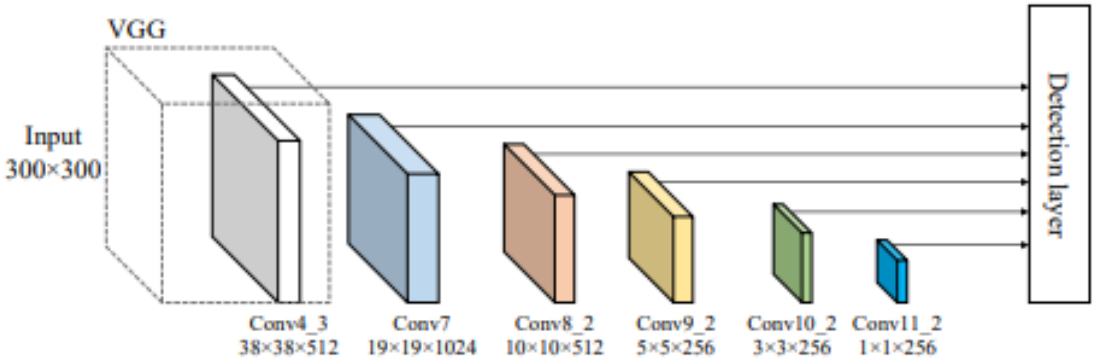


Figure 3.3: Block diagram of SSD with VGG16.

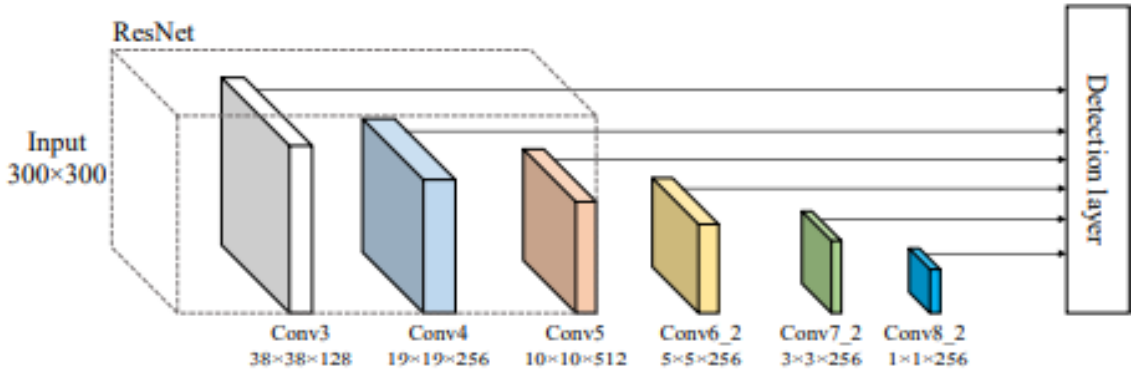


Figure 3.4: Block diagram of SSD with Resnet-50.

3.5 Training and Evaluation of the Network

For training and evaluation, a Tensor Processing Unit (TPU) and a virtual machine (VM) were used. Training and evaluation were done simultaneously to improve efficiency and reduce processing time.

Table 3.2: SSD model training parameters

Parameters
TFRecord files and pre-trained configuration file.
Anchor sizes: min 3, max 7.
Aspect ratios: 1, 2 and 0.5.
Type of activation: RELU.
Type of regularizer: L_2 Regularizer.
Kernel size: 3.
Classification loss algorithm: weighted sigmoid focal.
Localisation loss algorithm: weighted smooth L_1.
Training batch size: 64.
Training steps: 25,000.

TensorFlow library allows the continuous monitoring of the performance of the system while training and evaluation is continuing. This was a helpful feature that was used to evaluate the model's behaviour and possibly stop the training if parameters like error did not converge. Once training was done, the model was exported from the cloud to be used in a local machine. The script allowed one to choose the exactly the training step/epoch when the model was working as required.

Using a pre-trained network was chosen for this project and the network chosen had been trained on Microsoft Common Objects in Context (MSCOCO) dataset with 1.5 million objects (T. Lin et al., 2014). The weights from this Resnet-50 based SSD network was the starting point in the model (which is called a fine-tune checkpoint) and it cut down the training time from days to hours.

The training and model tuning configurations were set as per Table 3.2. The Table illustrates some of the parameters used to train the network as obtained from the authors of SSD network.

3.6 Robot Vision

USB (Universal Serial Bus) camera, HP 15 Mega-pixels resolution, was used as a vision sensor and was placed above the robot work table from a distance of 330 mm in the z axis. The height was chosen to enable the camera to have enough field of view of the platform where the objects were placed. Furthermore, this was the maximum height attainable by the robotic arm as per the forward kinematics. If the camera was placed a distance lower

or higher than 330 mm, it would make the images larger or smaller and would need an aspect ratio correction which would result in distortion of the images and poor detection accuracy. The distorted images would also pose a challenge for the robotic arm as it will not correctly identify the size of the object which would compromise grasping of objects. The camera was used by the neural network for object identification using colour, shape, and size.

Location information was also obtained from the sensor after the network had identified the object and *x* and *y* coordinate information obtained. The picking height of the objects was calculated, with 60 mm being an average of all the objects height. The location of the objects from the camera needed to be mapped before implementation of the inverse kinematic algorithm. Coordinate (0,0) was at the centre of the robot base and reference for all the objects' coordinates in the images. Figure. 3.5 illustrates the rectangular window of

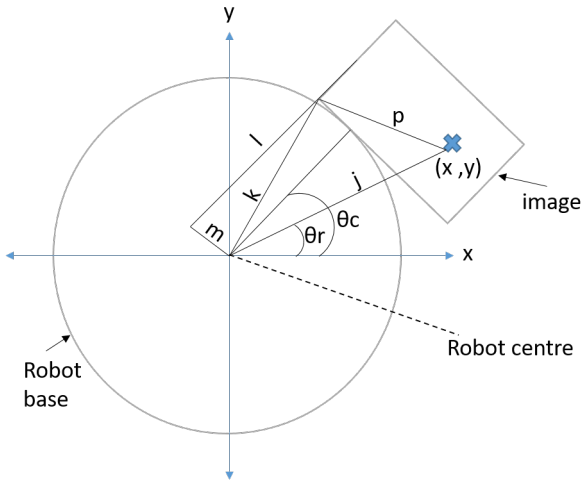
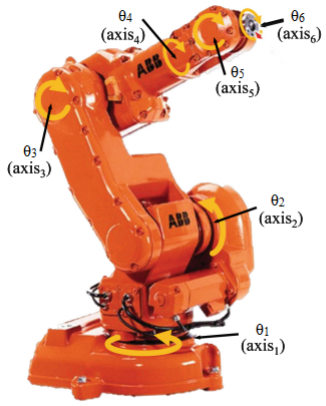


Figure 3.5: Mapping object location to centre of the base of the robotic arm.

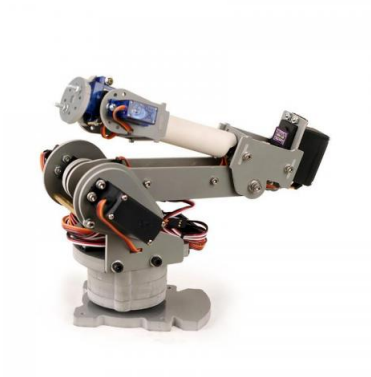
an image and the centre of the robot base while Equation. 3.1(Kazemi & Kharrati, 2017) describes how the angle of the object was determined.

$$\theta_r = \tan^{-1}\left(\frac{m}{l}\right) - \cos^{-1}\left(\frac{k^2 + j^2 - p^2}{2jk}\right) + \theta_c \tag{3.1}$$

where *m* is offset of the corner point of the image to the robot centre, *l* is the perpendicular distance from centre of the robot to the image, *k* is distance of the corner point of the image to robot centre, *j* is distance of the object to the robot centre, *p* is distance of the object to the image corner point, θ_c represents the angle the robot has moved from the *X*



(a) Representation of the IRB1400 robotic arm by ABB showing the degrees of freedom(Belda, 2018).



(b) Actual physical robotic arm used to run the experiments.

Figure 3.6: Images showing the original size and replica ARB1400 robotic arm used for the experiments.

axis and θ_r denotes the angle at which the image is being captured. The quantities m, l and θ_c (from the axis 1 servo motor position) are known, while k, j, and p can be determined using trigonometry. The location of the object with respect to the centre of the robotic arm can hence be obtained as

$$(x, y) = (j \cos\theta_r, j \sin\theta_r) \tag{3.2}$$

The z coordinate of the object is equal to the height of the camera which is 330 mm and remained constant throughout the experiments.

3.7 Robot Control

Figure 3.6 shows a 6 DOF model of miniature ABB IRB 1400 robotic arm that was used in this research. The arm had 6 analog servo motors at the joints giving the robot angular motion of up to 180 degrees. The gripper had one servo attached to enable closing and opening of the jaws. All the servo motors were controlled by a serial servo motor controller which was connected to a micro controller (Arduino mega) using UART protocol. The servo motors had an inbuilt PID control to ensure the servo reached the desired angle as fast as possible and accurately.

The flow chart of robot control is shown in Figure 3.7 where the different components of the system are synergistically combined. The camera took a picture of the platform with

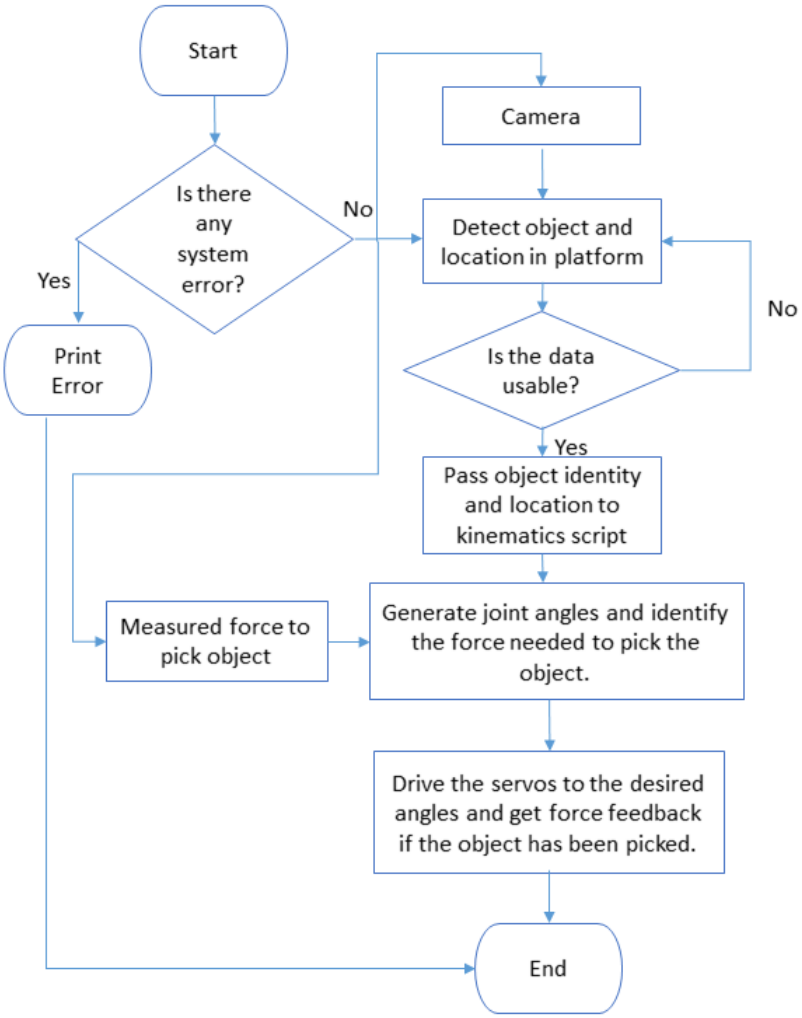


Figure 3.7: Flowchart of the steps followed from object detection to picking the object.

the object or objects under investigation. The image was processed by the neural network where detection was done and location information obtained. The location information was then passed to the inverse kinematic script that generated the joint angles and also identified the force needed depending on the object detected. The joint angles were then sent to the Arduino mega micro-controller using UART protocol where the angles were further assigned to their respective joint and the inbuilt PID controller of the servo motors ensured the set angles were achieved. The robot attempted grasping the object and with the

force sensor in the gripper, it was able to confirm picking and placing. Since the maximum force needed to pick an object had been pre-determined, a function in the micro-controller monitored the force from force sensor while picking to ensure the value was not exceeded which would have damaged the object or the gripper's servo motor.

3.8 Inverse Kinematics of IRB1400 Replica Robot

Inverse kinematics is the calculation of joint angles of a robotic arm given the coordinate of the end effector. For the IRB1400 model, the kinematic representation of the joints is shown in Figure 3.8. The axes were labelled from zero (shoulder) to six (gripper) with their respective rotation direction indicated along the axis that was allowed to rotate. The z axis was assigned to axis of rotation for the joint while x axis was freely assigned for the first joint. Other subsequent x axes pointed away from the previous joint. The y axis filled the remaining axis. The other important parameters considered included the distance between one axis to the next d and a being the offset distance from the axes.

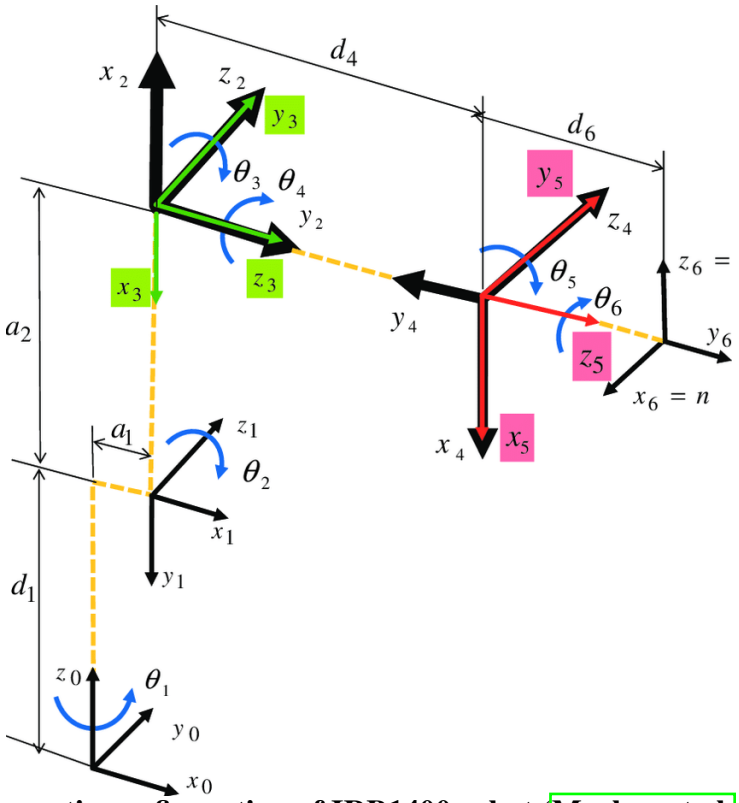


Figure 3.8: Kinematic configuration of IRB1400 robot (Markus et al., 2015).

Figure 3.8 was only used for referencing purposes but to generate Denavit-Hartenberg (DH) parameters for the robotic arm model, physical measurements were done and tabulated in Table 3.3. Since three consecutive axes of the arm intersect, Pieper's solution was used to compute the inverse kinematics(Serrezuela, Chavarro, Cardozo, Toquica, & Martinez, 2017). Before solving the inverse kinematics of a 6 DOF, it was assumed that the first three joints are entirely responsible for positioning the end effector, while any others are responsible for orientation. This assumption holds for robots with more than 3 DOF since the geometric method used for 3 or less DOF becomes very complicated to solve (Serrezuela et al., 2017). With the DH matrix computed, the *x*, *y* and *z* coordinates could be fed to the robotic arm inverse kinematic algorithm to generate joint angles θ_1, θ_2, θ_3, θ_4, θ_5 and θ_6. The values of *x* and *y* were obtained from the neural network bounding box. The object detected was wrapped around with a bounding box like shown in Figure 3.9 where two coordinates (x_1, y_1) and (x_2, y_2) of the diagonal of the box were picked and the mean of their length and coordinates of the centre of the length calculated. The height was held constant at 330 mm fro all objects hence the final coordinate was in three dimensions (*x, y,* and *z*).

3.9 Workspace Analysis

The work envelope of a robot is very important in determining the physical limits of the robotic arm. For the IRB1400 model, the DH parameters in Table 3.3 were measured physically on the robot and were used in MATLAB software to generate the workspace as shown in Figure 3.10. These parameters were used to populate all possible positions the end effector would reach, taking into consideration the joint angles and link lengths. The loop started at coordinate (0, 0, 0) and the inverse kinematics checked if the joint angles were reachable, if true, that point was marked as attainable by the robotic arm. If a coordinate was not attainable, it was not plotted. The maximum possible coordinates was attained when all the links were straight and parallel and when rotated, a sphere of

Table 3.3: DH Matrix To IRB-1400 Model Robot

Joint	1	2	3	4	5	6
d_i	40 mm	0 mm	26 mm	0 mm	0 mm	0 mm
a_i	110 mm	127 mm	26 mm	130 mm	0 mm	0 mm
α_i	90°	0°	90°	-90°	90°	0°
θ_i	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6

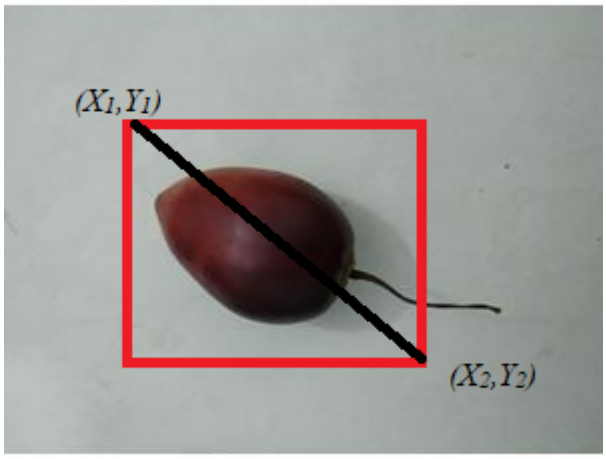


Figure 3.9: The coordinates of the mean length of the object to be picked.

maximum points was plotted. These points were used as the maximum limits and the last points that could be possibly plotted. The workspace was used to identify the working limits of the robot in all three axis as x:-0.3 m to 0.3 m, y: -0.2 m to 0.4 m and z: -0.1 m to 0.3 m.

3.10 Tactile Sensing

The robotic arm was designed with the ability to pick and place a wide range of objects and the amount of force needed to pick the objects was measured by using a Force Sensing Resistor (FSR) on the inner surface of the robot gripper. The FSR sensor chosen was from Sparkfun with a round sensing area of diameter 12.7 mm. The circular option was chosen over the square one so as to increase the sensing area.

Two options were available for sensing force, FSR sensor and a load cell. The FSR was chosen in place of load cell mainly because of the sensor's size and ability to be placed on the inner surface of the gripper. The smallest load cell examined was 8 mm thick as compared to 1 mm of the FSR sensor. The FSR sensor's resistance is inversely proportional to the force applied as shown in Figure 3.11 and the data sheet used to extract this plot is provide in Appendix III. The FSR was first calibrated using standard weights ranging from 100g to 1kg and there after a polynomial was obtained using *polyfit*, a MATLAB

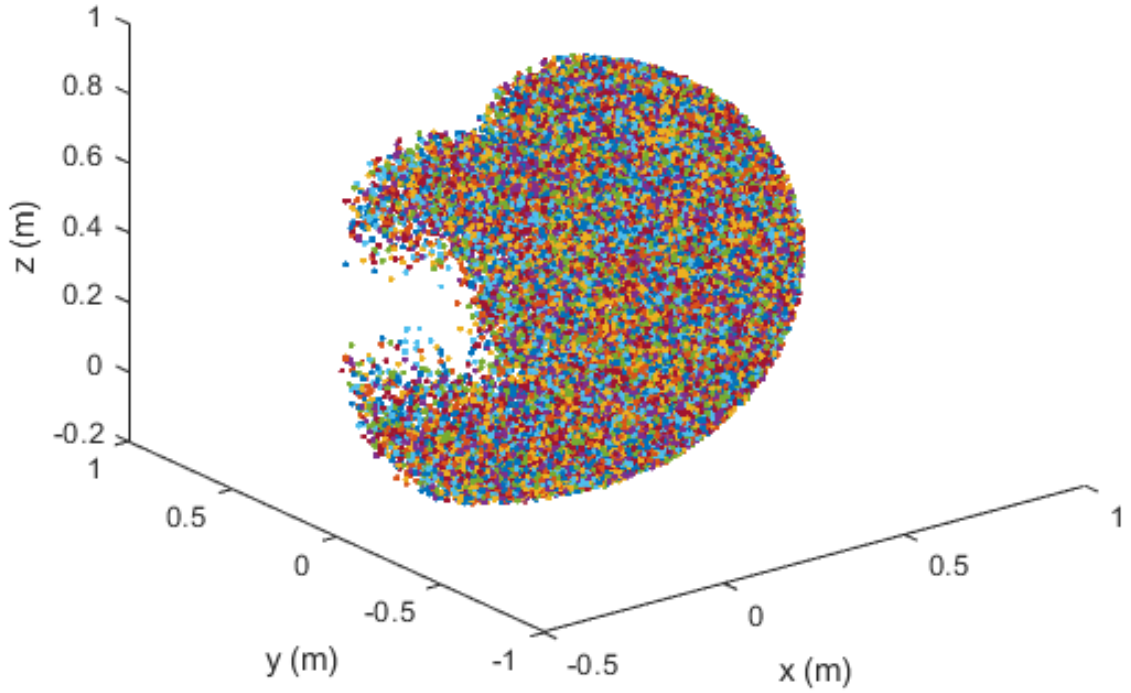


Figure 3.10: Sample workspace of the model ABB IRB1400 used in this experiment.

software function, that mapped the relationship between force and resistance. The polynomial was then used by Arduino Mega micro-controller to compute the force needed for grasping. The amount of force needed to grasp the objects for this experiment are depicted in Table 3.4.

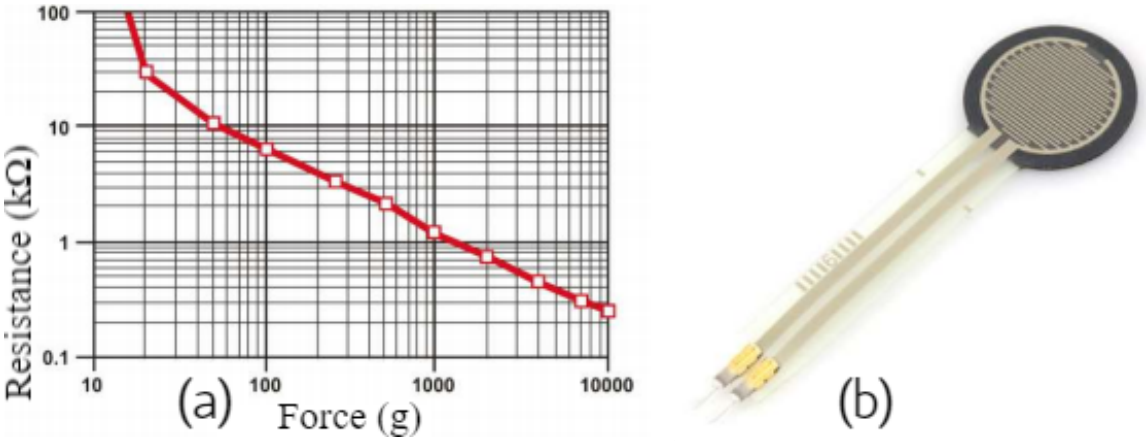


Figure 3.11: a: Graphical relationship between force applied versus resistance. b: FSR402 sensor.

Table 3.4: Maximum grasping force for individual objects.

Object	Grasping force (N)
Toothpaste	2.17
Toothbrush	1.97
Onion	1.97
Orange	1.96
Insulating tape	2.35
Phone charger	3.96
Remote control	3.72
Match box	2.60
Flash light	6.80
Tomato	1.99
Plastic toy	7.30
Pen	3.14
Tamarillo	1.96

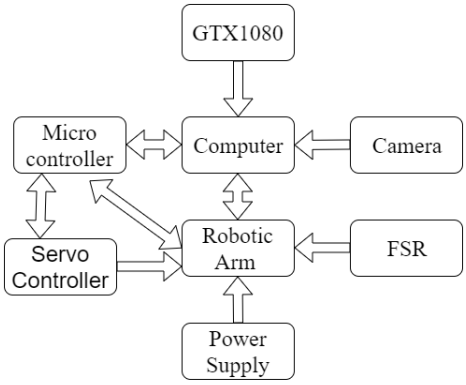


Figure 3.12: Block diagram of the system designed.

3.11 Experimental Setup

The block diagram of the setup used is shown in Figure 3.12 which includes a laptop with GTX1080 6GB graphics processor, a robotic arm with a gripper fitted with an FSR sensor, a microcontroller onto which the inverse kinematic algorithm is executed, a serial servo motor control to power the servo motors used at joints, a HP 15 Megapixel camera for object detection and some objects that were used for picking and placing. The actual setup used is depicted in Figure 3.13 while Figure 3.12 explained the flowchart. The inputs to the system were the camera, FSR, and power supply while the outputs were the servo motors in the robotic arm. The trained model was to be evaluated with the objects used to train it. This was achieved through placing the various objects on the platform and

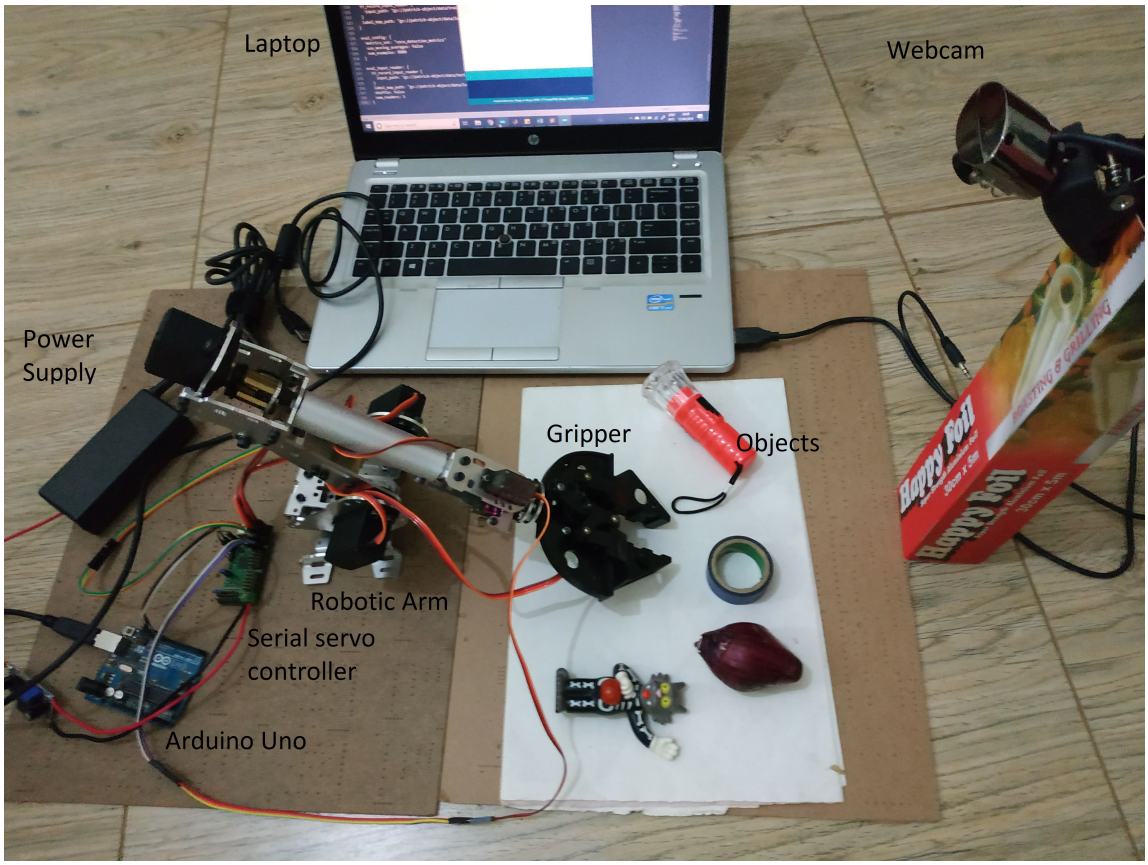


Figure 3.13: Actual experiment setup used.

the camera taking pictures of the objects. The images were then processed by the SSD network by first being fed through the Resnet-50 network for feature extraction. Feature extraction generates feature maps that are used to generate anchors of different sizes in height and width. The anchors are used to detect object of different sizes. The more the anchors the more detailed the description. The SSD network also generates multiscale feature blocks from the feature map to predict the class of the object and also draw the bounding boxes around the objects in the image. The network finally outputs the image, image class, object location and the detection accuracy in mAP.

To evaluate the accuracy of the neural network in detecting objects under different lighting conditions, two conditions were considered; using a 450 lm and a 1,600 lm lighting from a bulb. These bulbs were chosen to represent the minimum and maximum lighting in a medium scaled factory when artificial lighting and high intensity artificial lighting is used.

The objects were placed on the platform and the 450 lm bulb switched on. The accuracy of the neural network in detecting the objects was then generated for all objects. The same procedure was repeated with the 1,600 lm bulb and the values recorded.

To evaluate the accuracy of the network under partial occlusion, the objects were partially occluded up to 50 % while maintaining the camera above the objects. The images were then processed using TensorFlow and the detection accuracy was evaluated.

To evaluate the performance of the network under changing camera pose, the setup involved placing the camera directly overhead the platform and the objects placed below it. The camera then took images of the objects and the network gave out the accuracy in percentage form. The camera was then placed at 45° from the original position and the above steps repeated and the output data recorded. The angles between 0° and 45° were selected because it enabled the robotic arm to picking the objects without increasing the current drawn by the servo motors at the joints beyond a level that would damage them.

To evaluate real time speeds, the network was presented with three objects from the class used and the time it took for the network to correctly identify each object was measured with respect to the number of frames needed was noted. The value of time and number of frames was computed, averaged and tabulated in terms of frames per second.

Table 3.5 shows a summary of the experimental data setup. To measure the force needed to

Table 3.5: Experimental data setup

S/N	Parameter	Lower limit	Higher limit
1	Lighting conditions	450 lm	1600 lm
2	Occlusion	0 %	50 %
3	Camera position	0°	45°

grasp the objects, a force sensitive resistor was placed on the gripper's jaw and connected to the microcontroller. The servo motor connected to the gripper was closed in order to attempt grasping and once the object had been grasped, the values from the sensor were read. This was repeated for all the objects.

The system synergy was then evaluated from detection to placing. Objects were placed on the test area and the neural network instructed to detect the objects. After detection, the object location information was handled by the kinematics script that generated joint

angles for the robot and also the force needed to grasp the object. These values were passed to the micro controller through serial communication to execute the picking and finally placing the object at a designated location. The number of successful grasps, placing, and time taken were then recorded.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 SSD Performance Evaluation

The SSD network modelled using Resnet-50 was evaluated on TensorFlow software and parameters like precision, recall, learning rate, and intersection over union were graphed.

4.2 Precision Evaluation

The precision metric was used to evaluate the performance of the network. Precision measures the false positive rate or ratio of true object detections to the total number of objects that the network predicts (G. Wang, Hao, Ma, & Huang, 2010) and it is expressed as

$$PR = \frac{TP}{TP + FP} \tag{4.1}$$

where *TP* is true positive and *FP* is false positive.

The network was able to achieve greater than 0.98 mAP (98%) precision in detection of the objects that was trained on as shown in Figure 4.1. This high accuracy is attributed to the ResNet-50 feature extractor and transfer learning that was done prior to training the network to the custom dataset. The evaluated network precision in detecting small, medium and large object sizes are defined as follows

- Small objects: area <1024 square pixels.
- Medium objects: 1024 <area <9216 square pixels.
- Large objects: 9216 <area <1,000,000 square pixels.

The network hence had greater than 0.98 mAP (mean average precision) for large and medium sized objects. As per the figure, the precision for medium and large increased progressively as the training steps increased. This was as a result of the network learning distinguishing details about the objects used for training. The two plots of large and medium also followed each other throughout the training. This could be attributed to the fully connected layer in SSD-Resnet-50 architecture that allowed for accelerated learn-

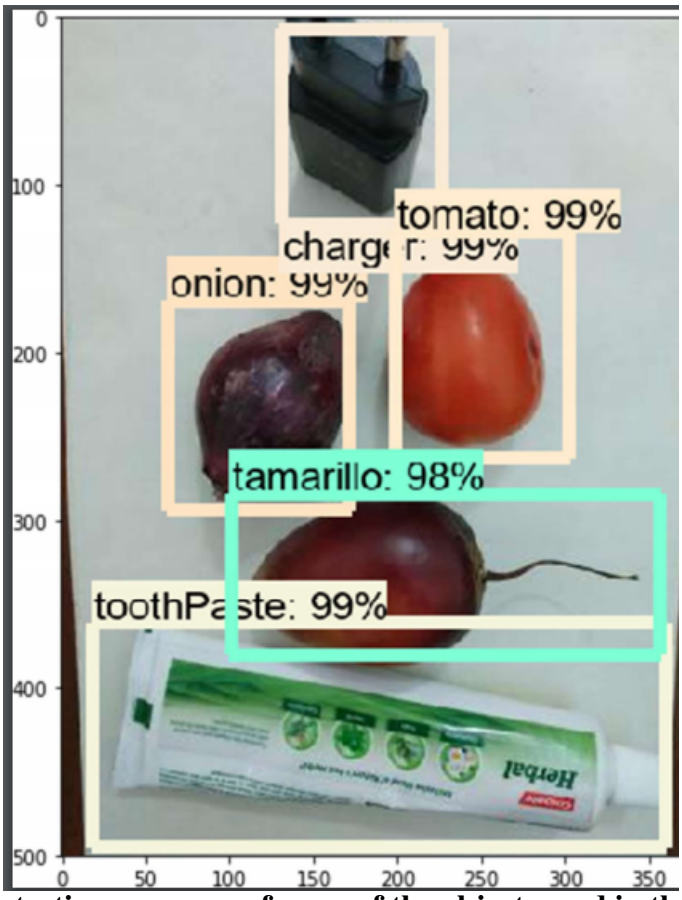


Figure 4.1: Detection accuracy of some of the objects used in the experiment.

ing of features in these two categories. These results show how good the network were at precisely identifying the objects in the 20% dataset that was reserved for testing. It was predicted that the network would achieve accuracies of above 0.9mAP which was attained. As shown in Figure 4.2, the network did not perform well with small objects mainly because the images used to train the network did not have small objects. The pictures of the objects were taken at a height of 330mm in the Z axis axis which meant that all images of the objects had pixels greater than 1024 and hence the category of medium and large for all pictures. The height was also chosen because it was within a range that allowed for accurate picking of the objects by the robot. If the camera was placed higher, issues of aspect ratio and distortion of the image would result in errors in object location identification.

The robot's links were present in some of the images and since the links were rectangular in shape, the network identified them as remote controllers. This resulted in the network

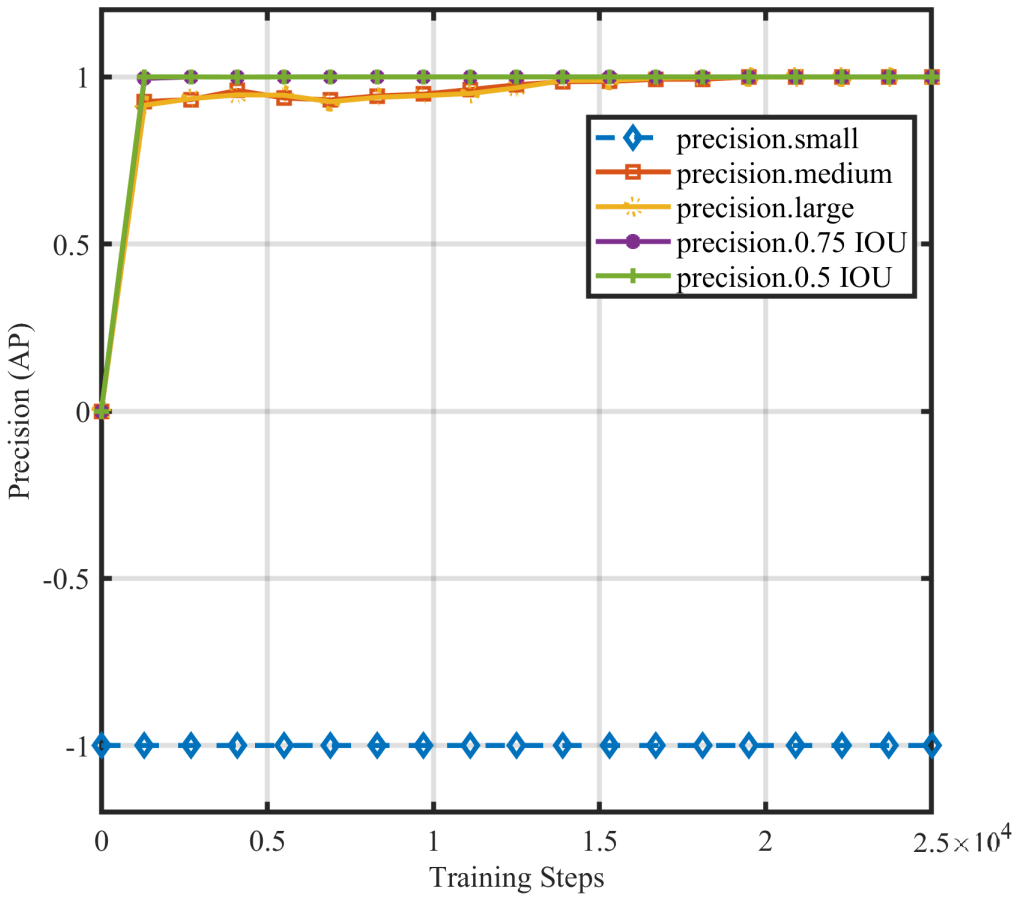


Figure 4.2: Precision of the network when detecting objects.

displaying an extra object, remote control, which was undesired. This was rectified by placing the robotic arm in a home position that was out of range of the field of view of the camera.

4.3 Intersection Over Union (IOU)

The network was also evaluated with respect to the precision of placing bounding boxes over the desired object using a method called intersection over union (IOU). The IOU uses ground truth bounding box that was generated when the images were being annotated and the predicted bounding box. The two boxes are overlaid over each other and the IOU generated. A high IOU means the boxes generated are placed exactly where they are supposed to while a lower IOU doesn't fit or it overlaps over a small region. The two parameters used for the network were 50% and 75% IOU since they gave a more detailed

evaluation of the network. 50%-IOU showed the ability of the network to draw a bounding box around an object precisely more than 50% of the time. 75%-IOU showed the same performance but the precision levels were increased. As shown in Figure 4.2, the network was able to achieve 100% precision intersections after 25,000 training steps. This high accuracy can be attributed to the use of SSD with Resnet-50 which as presented earlier has better accuracy.

4.4 Recall Rate

The recall rate of the network was also analysed. As noted in (G. Wang et al., 2010), recall rate is defined as a measure of false negative rate or the ratio of true object detection to the total number of objects in the dataset and is expressed as shown in Equation. 4.2.

$$RC = \frac{TP}{TP + FN} \tag{4.2}$$

where *TP* is true positive and *FN* is false negative.

A value of 1.0 mAP is a likelihood of the network to positively detect all the objects in the dataset. . As shown in Figure 4.3, the network recall rate was above 0.85 mAP after 6,000 training steps and by the end, 25000, the network could recall 1, 10 and 100 detections per image with accuracies of above 0.90 mAP except for small objects. This result means that the network was able to detect 1, 10 or 100 objects in an image with an accuracy of 0.90 mAP if the same dataset was used. This is important because it shows how robust the network is when handling a large set of objects in a single image. The same challenges of processing small objects were also exhibited here as the network was completely not able to recall small objects hence a value of -1 mAP. The number of multiscale blocks generated for these images were not sufficient to increase accuracy of the network with small objects.

4.5 Learning Rate

The learning rate is a measure of how fast the weights of the neural network are being adjusted to achieve learning. The rate is initially very fast at 0.04 but slows down to almost zero as it tends to a local minimum or the best accuracy for the network. The curve follows what is expected of learning rate using Gradient Descent training optimising

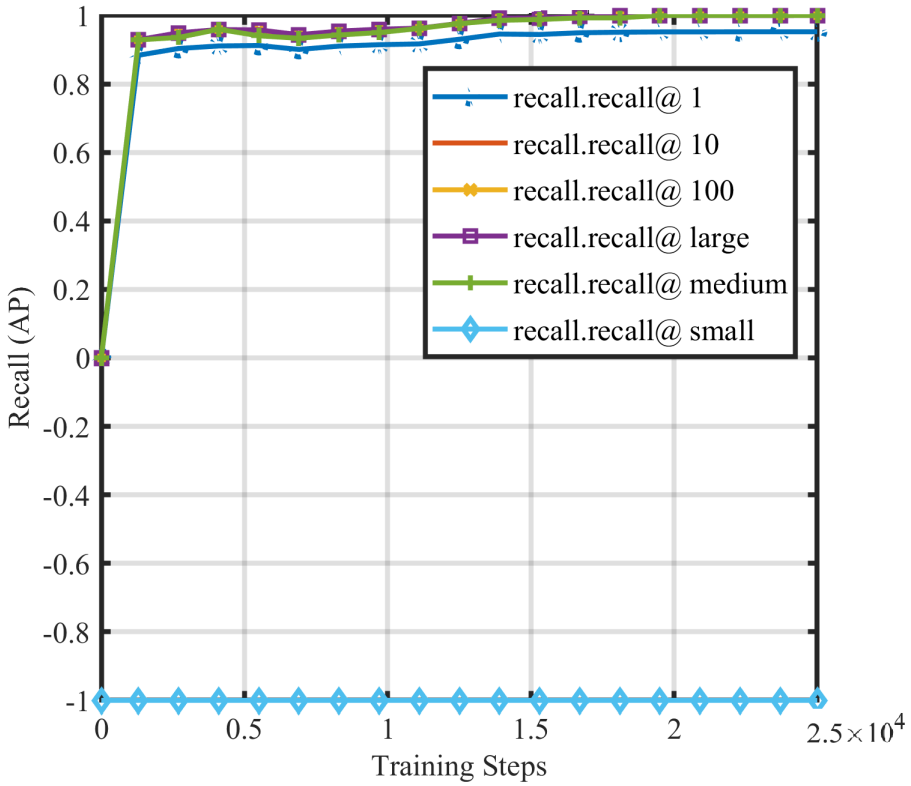


Figure 4.3: Recall rate of the network.

algorithm where as the training starts, the error is large and the algorithm tries to minimize this error at every iteration. Over time, the error is minimized and the epoch closest to zero is picked to stop the training.

The network hit peak learning rate at the 4000 step and started dropping off until the training was halted as depicted in Figure 4.4. The rate rises up first because the rate that was declared initially was too low and the training process increased it to a level it could be efficient. The rate peaked because past 0.04, the network wouldn't be able to detect any local minima and it would lead to errors in detection. The peak was at the 4000 step since it followed gradient descent and this was the point at which the total-loss had reduced by more than 80% from the starting point (at 0 step total-loss was at 2.5 and at 4000 it was at 0.45). The use of Tensor Processing Units (TPUs) can also be credited with the fast learning rate because the processor has been optimized to achieve this and it outperforms graphic processing units (GPUs). The network peaks at

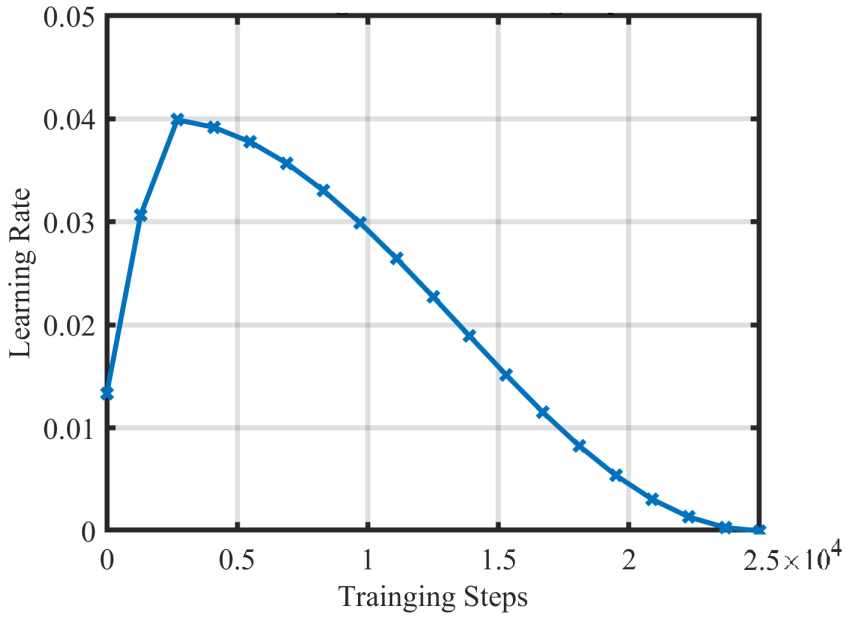


Figure 4.4: Learning rate of the network throughout the training steps.

4.6 Network's Loss

Loss parameters like regularisation, classification, localisation and total loss for a network during training represent the cost of inaccuracies. These costs should reduce with time and approach zero for a very accurate training. As depicted in Figure 4.5, the network's classification was very accurate in classifying objects in images correctly and it had levelled out at almost zero by 12500 steps. The localisation error depicted in Figure 4.5, shows how accurate the network is at locating the objects in images. The loss is close to zero at the early stages of training meaning the network had learnt the objects and the loss plateaued at this level until the end of the training; hence, maintaining the learnt accuracies.

Regularisation is a technique used to discourage complexities in the trained network. These complexities arise when the size of the training parameters increase leading to overfitting which is not desirable. To discourage complexities, regularisation penalizes the loss function as shown in Figure 4.5. The regularisation loss decreases progressively towards zero with the increase in training steps. By the end of the training, the model can be declared not to be suffering from overfitting. Total loss is the summation of all these losses and it gives a picture of all the losses with respect to the model's performance.

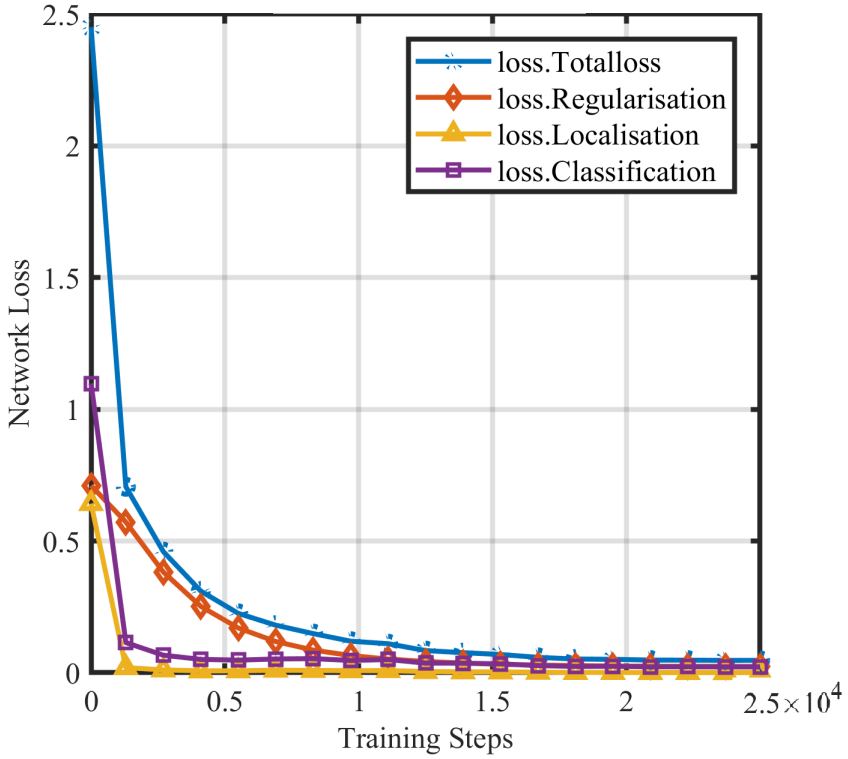


Figure 4.5: The loss graphs for the network while training.

4.7 SSD Performance under Different Lighting Conditions

The network was also tested on different lighting conditions using bulbs of different lumens and it was still able to achieve above 0.90 mAP under both conditions as shown in Table 4.1. The Table shows the different set of experiment results ranging from how the network performed when exposed to different lighting conditions, partially occluded, different camera pose, and number of successful grasps by the robotic arm (the system had three chances to pick an object under 1600 lm lighting and if it was successful once it was recorded as 1/3).

The percentage values in Table 4.1 with respect to the objects were obtained as follows, using an onion as an example: The first experiment was done with a 1600 lm bulb and the following subsequent experiments done using a 450 lm bulb. The onion was first placed in the working area and a bulb of 1600 lm was switched on, the object detection python script was activated and the camera identified the onion. The script then returned the precision on detection in percentage for the object under 1600 lm lighting. The experiment was

repeated with a 450 lm bulb and the results tabulated. The onion was blocked by 50% using a piece of white paper and the object detection script was again run and the results noted. The camera angle was then adjusted to 45° from the top and the procedure repeated. Finally, the pick-and-place algorithm was activated and the robot instructed to pick the onion from the working platform and place it in a designated position. This was done three times and every attempt was recorded.

These results confirm that the network developed is robust enough to be used in varied lighting conditions without a huge loss in detection accuracies and overall robotic pick and place application.

4.8 Occlusion

Occlusion is a challenge for object detection and a well trained network should be able to distinguish specific objects and provide grasping information for the robotic arm. The trained network was tested with objects that were occluded by approximately 50% and the network was able to achieve above 75% accuracy as depicted in Table 4.1. This confirmed that the network can be used in real life application but with slight improvement to increase on its accuracies.

Table 4.1: Performance of object detection and pick-and-place system

Object	450 lm bulb	1600 lm bulb	50% occlusion (450 lm)	45° pose (450 lm)	Successful grasps (450 lm)
	(%)	(%)	(%)	(%)	
Toothpaste	92	99	85	82	2/3
Toothbrush	92	97	83	70	1/3
Onion	90	98	89	83	3/3
Orange	91	96	90	85	3/3
Insulating tape	92	97	75	75	2/3
Phone charger	93	99	89	78	1/3
Remote control	91	98	88	69	1/3
Match box	93	99	87	82	3/3
Flash light	95	100	86	77	2/3
Tomato	93	96	87	89	3/3
Plastic toy	91	100	90	78	1/3
Pen	92	99	88	83	1/3
Tamarillo	91	98	86	88	2/3

Table 4.2: Real time speeds versus pixel area

Pixels	Developed network speed (fps)	Original SSD speed (fps)
300 x 300	-	46
400 x 400	40	-
500 x 500	25	19

4.9 Camera pose

The change in camera pose demonstrates how versatile a network is when viewing the object from a different angle other than the one that the network was trained on. Table 4.1 illustrates the impact of changing the viewing angle of the network by 45° and it follows that accuracy decreased but to a minimum of 0.69 mAP which was obtained when evaluating the remote control. This depicted that the network was versatile enough to work other different camera poses. These results showed that the network could to be incorporated into a different camera mounting configuration other than the fixed-camera configuration used while still delivering accuracy of above 0.60 mAP.

4.10 Real Time Speed Evaluation

The SSD with a Resnet-50 backbone was able to achieve real time speeds of 40 fps when the input image was restricted to 400×400 pixels. When the pixels was increased to 500×500 pixels, the speed dropped to 25 fps. These values are slightly better than the original work done by the developers of SSD (Liu et al., 2016) who got a speed of 46fps for 300×300 pixels and 19 fps for 500×500 pixels as depicted in Table 4.2.

4.11 Grasping

The force needed to grasp the objects was very crucial to achieve successful picking and placing. Table 3.4 shows the force sensor readings needed to grasp each object. With the force sensor values, the final experiment was conducted. The robot made three attempts to detect, pick and place each object and the results were tabulated in Table 4.1. The picking and placing took 15-30 seconds depending on the object being picked. The majority of this time was spent on moving the arm from home position towards the object, picking the object, taking the object a designated location, placing and finally going back to home position. Each servo motor was moving at 0.16 sec per degree at 6V for every joint. Objects that performed well were those once with symmetrical bodies and also those ones

made from shapes like spheres and rectangles as can be seen by objects like tamarillo, tomato, match box, orange, and onion.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this work, a problem of using a pick and place robotic arm in conditions of uncertainty like varying lighting, change in camera pose, and partial occlusion while attaining real time speeds was tackled SSD Resnet-50 Convolutional Neural Network was chosen for the task.

Attaining real time speeds was a major point in this work because it had a direct relation to how fast a pick and place robotic arm could perform. The uncertainties mimic real world conditions experienced in the industry. The system developed was able to achieve real time object object detection at speeds of 40 fps with accuracies of over 75% under conditions of uncertain lighting, change in camera pose and partial occlusion. The overall system robust enough to pick objects with satisfactory grasping accuracies meeting the objectives set out.

It was determined that introducing a backbone network in pre-existing CNN networks can improve the performance of an object detection network. With proper application of the network, other improvements can be achieved like increasing the efficiency of a pick and place robotic arm. From the experiments, it was proven that the fixed camera configuration can be replaced with a more versatile and efficient eye-in-hand configuration. This configuration allows the camera to be placed in the gripper and since the network is still able to detect objects at other angles, manoeuvrability of the robotic arm could be increased. This increase means that robot can pick objects more faster than the existing fixed camera configuration.

This was proof that the system developed could be used in industry and could compete with existing systems. Also, this work shows how powerful and flexible Convolution Neural Networks have become in solving real world problems especially in the field of robotics.

5.2 Recommendations for Future Work

The workspace of the robotic arm was greatly limited because of the 180° servo motors used at the joints. The 360° motors that were available all had plastic gearing which were not strong enough to supply the torque demands of the joints since they kept wearing out. The metallic version of the same motors were expensive and did not meet the budget requirements. This meant some positions calculated by the inverse kinematic algorithm couldn't be attained and accuracy was compromised. The use of RGB camera was effective in object detection but it was not able to detect the distance between the object and the end effector. This limited the system to using a fixed camera configuration as opposed to camera in hand configuration which is more accurate. Fixed camera configuration has a large field of view but the robot's links appear in the images captured which is factored as noise. For real time robots, such noises reduce accuracy and eye-in-hand configuration is hence preferred. Introduce a distance camera and installing 360° servo motors to the arm will improve the performance of the pick and place robot since distance can be obtained very fast with little effort.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th usenix conference on operating systems design and implementation* (pp. 265–283). Berkeley, CA: USENIX Association.

Akogo, D. A., & Palmer, X.-L. (2019). Scaffoldnet: Detecting and classifying biomedical polymer-based scaffolds via a convolutional neural network. In *Future of information and communication conference* (pp. 152–161). San Francisco.

Arbeláez, P., Pont-Tuset, J., Barron, J. T., Marques, F., & Malik, J. (2014). Multiscale combinatorial grouping. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 328–335). Columbus, Ohio.

Belda, K. (2018). Nonlinear design of model predictive control adapted for industrial articulated robots. In *Icinco (2)* (pp. 81–90). Porto, Portugal.

Bell, S., Lawrence Zitnick, C., Bala, K., & Girshick, R. (2016). Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2874–2883). Las Vegas, Nevada.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of international conference on computational statistics (compstat)* (pp. 177–186). Paris, France.

Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., & Rother, C. (2014). Learning 6d object pose estimation using 3d object coordinates. In *European conference on computer vision* (pp. 536–551). Zurich, Switzerland.

Chen, S., Liu, J., Niu, W., Han, Y., Xiao, Y., Xue, Y., & Ye, X. (2019). Research on the detection method for insulation piercing connectors and bolts on the transmission line based on ssd algorithm. In *2019 ieee 4th advanced information technology, electronic and automation control conference (iaeac)* (Vol. 1, pp. 960–964). Chengdu, China.

Chen, W., Qu, T., Zhou, Y., Weng, K., Wang, G., & Fu, G. (2014). Door recognition and deep learning algorithm for visual based robot navigation. In *2014 ieee international conference on robotics and biomimetics (robio 2014)* (pp. 1793–1798). Bali, Indonesia.

Ciaburro, G., Ayyadevara, V. K., & Perrier, A. (2018). *Hands-on machine learning on*

google cloud platform: Implementing smart and efficient analytics using cloud ml engine. Packt Publishing.

Cilimkovic, M. (2015). Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*, *15*.

Dai, J., Li, Y., He, K., & Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems* (pp. 379–387).

De Magistris, G., Munawar, A., & Vinayavekhin, P. (2016, December). Teaching a robot pick and place task using recurrent neural network. In *ViEW2016*. Yokohama, Japan.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2010). Imagenet: A large-scale hierarchical image database. In *2010 ieee conference on computer vision and pattern recognition* (pp. 248–255). Miami Beach, FL.

DeTone, D., Malisiewicz, T., & Rabinovich, A. (2018). Superpoint: Self-supervised interest point detection and description. In *Proceedings of the ieee conference on computer vision and pattern recognition workshops* (pp. 224–236). Salt lake city, Utah.

Dobbin, K. K., & Simon, R. M. (2011). Optimally splitting cases for training and testing high dimensional classifiers. *BMC medical genomics*, *4*(1), 31.

Dusmanu, M., Rocco, I., Pajdla, T., Pollefeys, M., Sivic, J., Torii, A., & Sattler, T. (2019). D2-net: A trainable cnn for joint description and detection of local features. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 8092–8101). Seoul, Korea.

Erhan, D., Szegedy, C., Toshev, A., & Anguelov, D. (2014). Scalable object detection using deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2147–2154). Columbus, Ohio.

et al, A. Z. (2017). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *Computing Research Repository (CoRR)*, *abs/1710.01330*.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2011). The pascal visual object classes (voc) challenge. *International journal of computer vision*, *88*(2), 303–338.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2011). Object detection with discriminatively trained part-based models. *IEEE transactions on*

pattern analysis and machine intelligence, *32*(9), 1627–1645.

Ferdinando, H., Wicaksono, H., & Wibowo, R. (2017). The implementation of pid controller in the pick and place robot. *Dept. of Electrical Engineering, Petra Christian University, Indonesia*.

Freund, Y., & Schapire, R. E. (2015). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (pp. 23–37). Barcelona, Spain.

Fu, C.-Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. C. (2017). Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 1440–1448). Las Condes, Chile.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 580–587). Columbus, Ohio.

Gonzalez, A., Villalonga, G., Ros, G., Vázquez, D., & López, A. M. (2015). 3d-guided multiscale sliding window for pedestrian detection. In *Iberian conference on pattern recognition and image analysis* (pp. 560–568). Santiago de Compostela, Spain.

Gu, W., Xiong, Z., & Wan, W. (2013). Autonomous seam acquisition and tracking system for multi-pass welding based on vision sensor. *The international journal of advanced manufacturing technology*, *69*(1-4), 451–460.

Gupta, S., Girshick, R., Arbeláez, P., & Malik, J. (2014). Learning rich features from rgb-d images for object detection and segmentation. In *European conference on computer vision* (pp. 345–360). Zurich, Switzerland.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, *37*(9), 1904–1916.

Hinton, G. E., Krizhevsky, A., & Wang, S. D. (2011). Transforming auto-encoders. In *International conference on artificial neural networks* (pp. 44–51). Espoo, Finland.

Hossain, D., & Capi, G. (2016, March). Application of deep belief neural network for robot object recognition and grasping. In *The 2nd ieej international workshop on sensing, actuation, and motion control (samcon 2016).* Tokyo, Japan.

Houska, B., Li, J. C., & Chachuat, B. (2018). Towards rigorous robust optimal control via generalized high-order moment expansion. *Optimal Control Applications and*

Methods, *39*(2), 489–502.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... others (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 7310–7311). Honolulu, Hawaii.

Huang, P.-C., & Mok, A. K. (2018). A case study of cyber-physical system design: Autonomous pick-and-place robot. In *2018 ieee 24th international conference on embedded and real-time computing systems and applications (rtcsa)* (pp. 22–31). Hakodate, Japan.

Javadi, M. H. M., Dolatabadi, H. R., Nourbakhsh, M., Poursaeedi, A., & Asadollahi, A. R. (2012). An analysis of factors affecting on online shopping behavior of consumers. *International Journal of Marketing Studies*, *4*(5), 81.

Juang, C.-F., & Chen, G.-C. (2011). A ts fuzzy system learned through a support vector machine in principal component space for real-time object detection. *IEEE Transactions on Industrial Electronics*, *59*(8), 3309–3320.

Karabegović, I., Karabegović, E., Mahmić, M., & Husak, E. (2015). The application of service robots for logistics in manufacturing processes. *Advances in Production Engineering & Management*, *10*(4).

Kazemi, S., & Kharrati, H. (2017, Mar 01). Visual processing and classification of items on moving conveyor with pick and place robot using plc. *Intelligent Industrial Systems*, *3*(1), 15–21. Retrieved from https://doi.org/10.1007/s40903-017-0071-3 doi: 10.1007/s40903-017-0071-3

Kondo, N. (2010). Automation on fruit and vegetable grading system and food traceability. *Trends in Food Science & Technology*, *21*(3), 145–152.

Konrad Ahlin, A. H., Benjamin Joffe, & McMurray, G. (2016). Autonomous leaf picking using deep learning and visual-servoing. *International Federation of Automatic Control*, *49-16*, 177-183.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 1097–1105). Curran Associates, Inc.

Kucuk, S., & Bingul, Z. (2011). Robot kinematics: Forward and inverse kinematics. In *Industrial robotics: Theory, modelling and control.* IntechOpen.

Kuipers, J. B., et al. (2009). *Quaternions and rotation sequences* (Vol. 66). Princeton

university press Princeton.

Kumar, R., Lal, S., Kumar, S., & Chand, P. (2014). Object detection and recognition for a pick and place robot. In *Asia-pacific world congress on computer science and engineering* (p. 1-7). Nadi, Fiji. doi: 10.1109/APWCCSE.2014.7053853

Kuo, W., Hariharan, B., & Malik, J. (2015). Deepbox: Learning objectness with convolutional networks. In *Proceedings of the ieee international conference on computer vision* (pp. 2479–2487). Santiago, Chile.

Lazebnik, S., Schmid, C., & Ponce, J. (2010). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2010 ieee computer society conference on computer vision and pattern recognition (cvpr'10)* (Vol. 2, pp. 2169–2178). New York, USA.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lee, C., Kim, H. J., & Oh, K. W. (2016). Comparison of faster r-cnn models for object detection. In *2016 16th international conference on control, automation and systems (iccas)* (pp. 107–110). Gyeongju, Korea.

Lenz, I., Lee, H., & Saxena, A. (2015). Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, *34*(4-5), 705–724.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., & Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, *37*(4-5), 421–436.

Li, Y., & Yuan, Y. (2017). Convergence analysis of two-layer neural networks with relu activation. *arXiv preprint arXiv:1705.09886*.

Lin, H. I., Chen, Y. Y., & Chen, Y. Y. (2015, May). Robot vision to recognize both object and rotation for robot pick-and-place operation. In *2015 international conference on advanced robotics and intelligent systems (aris)* (p. 1-6). Taipei, Taiwan. doi: 10.1109/ARIS.2015.7158364

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., ... Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *Computing Research Repository (CoRR)*, *abs/1405.0312*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21–37). Amsterdam, The Netherlands.

Mariolis, I., Peleka, G., Kargakos, A., & Malassiotis, S. (2015). Pose and category recognition of highly deformable objects using deep learning. In *2015 international*

conference on advanced robotics (icar) (pp. 655–662). Istanbul, Turkey.

Markus, E., Agee, J., Jimoh, A.-G., & Hamam, Y. (2015). *Coordinated control of multiple robotic manipulators based on differential flatness* (Doctoral dissertation, TSHWANE UNIVERSITY OF TECHNOLOGY). doi: 10.13140/RG.2.2.16137.95842

Neverova, N., Wolf, C., Taylor, G. W., & Nebout, F. (2014). Multi-scale deep learning for gesture detection and localization. In *European conference on computer vision* (pp. 474–490). Zurich, Switzerland.

Ono, Y., Trulls, E., Fua, P., & Yi, K. M. (2018). Lf-net: Learning local features from images. *arXiv preprint arXiv:1805.09662*.

Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1717–1724). Columbus, Ohio.

Patin, F. (2013). An introduction to digital image processing. *online]: http://www. programmersheaven. com/articles/patin/ImageProc. pdf*.

Perronnin, F., Sánchez, J., & Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In *European conference on computer vision* (pp. 143–156). Heraklion, Crete, Greece.

Pierson, H. A., & Gashler, M. S. (2017). Deep learning in robotics: a review of recent research. *Advanced Robotics*, *31*(16), 821–835.

Pinaya, W. H. L., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Convolutional neural networks. In *Machine learning* (pp. 173–191). Elsevier.

Pinheiro, P. O., Lin, T.-Y., Collobert, R., & Dollár, P. (2016). Learning to refine object segments. In *European conference on computer vision* (pp. 75–91). Amsterdam, The Netherlands.

Punjani, A., & Abbeel, P. (2015). Deep learning helicopter dynamics models. In *2015 ieee international conference on robotics and automation (icra)* (pp. 3223–3230). Seattle, Washington.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016, Jun). You only look once: Unified, real-time object detection. Las Vegas, Nevada. doi: 10.1109/cvpr.2016.91

Redmon, J., & Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 7263–7271). Honolulu, Hawaii.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Ren, X., & Ramanan, D. (2013). Histograms of sparse codes for object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3246–3253). Portland, Oregon.

Rothe, R., Guillaumin, M., & Van Gool, L. (2014). Non-maximum suppression for object detection by passing messages between windows. In *Asian conference on computer vision* (pp. 290–306). Singapore.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Serrezuela, R. R., Chavarro, A. F. C., Cardozo, M. A. T., Toquica, A. L., & Martinez, L. F. O. (2017). Kinematic modelling of a robotic arm manipulator using matlab. *ARPN Journal of Engineering and Applied Sciences*, *12*(7), 2037–2045.

Shah, S., Saha, S. K., & Dutt, J. K. (2009). Denavit-hartenberg parameters of euler-angle-joints for order (n) recursive forward dynamics. In *Asme 2009 international design engineering technical conferences and computers and information in engineering conference* (pp. 115–120). San Diego, California.

Shen, Z., Liu, Z., Li, J., Jiang, Y.-G., Chen, Y., & Xue, X. (2017). Dsod: Learning deeply supervised object detectors from scratch. In *Proceedings of the ieee international conference on computer vision* (pp. 1919–1927). Venice, Italy.

Smirnov, E. A., Timoshenko, D. M., & Andrianov, S. N. (2014). Comparison of regularization methods for imagenet classification with deep convolutional neural networks. *2nd AASRI Conference on Computational Intelligence and Bioinformatics*, *6*(Supplement C), 89 - 94.

Taylor, G. W., Spiro, I., Bregler, C., & Fergus, R. (2011). Learning invariance through imitation. In *Conference on computer vision and pattern recogition 2011* (pp. 2729–2736). Colorado, USA.

Tzutalin. (2015). *Labelimg*. Free Software: MIT License. Retrieved from `https://github.com/tzutalin/labelImg`

Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, *104*(2), 154–171.

Viola, P., & Jones, M. (2011). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2011 ieee computer society conference on computer vision and pattern recognition. cvpr 2011* (Vol. 1, p. I-511). Kauai, Hawaii. doi: 10.1109/CVPR.2001.990517

Wang, G., Hao, J., Ma, J., & Huang, L. (2010). A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert systems with applications*, *37*(9), 6225–6232.

Wang, X., Qin, Y., Wang, Y., Xiang, S., & Chen, H. (2019). Reltanh: An activation function with vanishing gradient resistance for sae-based dnns and its application to rotating machinery fault diagnosis. *Neurocomputing*, *363*, 88–98.

Wei, G., Li, G., Zhao, J., & He, A. (2019). Development of a lenet-5 gas identification cnn structure for electronic noses. *Sensors*, *19*(1), 217.

Xiong, Z., Yao, Z., Ma, Y., & Wu, X. (2019). Vikingdet: A real-time person and face detector for surveillance cameras. In *2019 16th ieee international conference on advanced video and signal based surveillance (avss)* (pp. 1–7). Taipei, Taiwan.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Xue, J., Li, J., & Gong, Y. (2013). Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech* (pp. 2365–2369).

Yang, Y., Li, Y., Fermuller, C., & Aloimonos, Y. (2015). Robot learning manipulation action plans by" watching" unconstrained videos from the world wide web. In *Twenty-ninth aaai conference on artificial intelligence.* Austin, Texas.

Yoo, H.-J. (2015). Deep convolution neural networks in computer vision: a review. *IEIE Transactions on Smart Processing & Computing*, *4*(1), 35–43.

Zhang, Y., Sohn, K., Villegas, R., Pan, G., & Lee, H. (2015). Improving object detection with deep convolutional networks via bayesian optimization and structured prediction. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 249–258). Boston, Massachusetts.

Zhao, Z.-Q., Zheng, P., Xu, S.-t., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), 3212–3232.

Zitnick, C. L., & Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *European conference on computer vision* (pp. 391–405). Zurich, Switzerland.

APPENDIX I

SOFTWARE

I.1 Object Detection Python Code

```
import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile

import tensorflow as tf

import zipfile


from collections import defaultdict

from io import StringIO

from matplotlib import pyplot as plt

from PIL import Image
```

```
from utils import label_map_util

from utils import visualization_utils as vis_util

# What model to download.
MODEL_NAME = 'output_1'

#MODEL_FILE = MODEL_NAME + '.tar.gz'

#DOWNLOAD_BASE =

'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model

 that is used for the object detection.

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct

label for each box.
PATH_TO_LABELS = os.path.join('data', 'label-map.pbtxt')

NUM_CLASSES = 13


# ## Download Model

#if not os.path.exists(MODEL_NAME + '/frozen_inference_graph.pb'):
#        print ('Downloading the model')
#        opener = urllib.request.URLopener()
#        opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
#        tar_file = tarfile.open(MODEL_FILE)
```

```
#        for file in tar_file.getmembers():
#          file_name = os.path.basename(file.name)
#          if 'frozen_inference_graph.pb' in file_name:
#            tar_file.extract(file, os.getcwd())
#        print ('Download complete')
#else:
#        print ('Model already exists')

# ## Load a (frozen) Tensorflow model into memory.

detection_graph = tf.Graph()
with detection_graph.as_default():
od_graph_def = tf.GraphDef()
with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
serialized_graph = fid.read()
od_graph_def.ParseFromString(serialized_graph)
tf.import_graph_def(od_graph_def, name='')


# ## Loading label map
# Label maps map indices to category names,

 so that when our convolution network

 predicts `5`, we know that this corresponds

 to `airplane`.  Here we use internal utility

  functions, but anything that returns a

   dictionary mapping integers to appropriate

    string labels would be fine
```

```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories

(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

#intializing the web camera device

import cv2
cap = cv2.VideoCapture(0)

# Running the tensorflow session
with detection_graph.as_default():
with tf.Session(graph=detection_graph) as sess:
ret = True
while (ret):
ret,image_np = cap.read()
# Expand dimensions since the model expects

 images to have shape:
[1, None, None, 3]
image_np_expanded = np.expand_dims(image_np, axis=0)

image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
# Each box represents a part of the image

where a particular object was detected.
boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
# Each score represent how level of

confidence for each of the objects.
# Score is shown on the result image, together with the

 class label.
```

```
scores = detection_graph.get_tensor_by_name

('detection_scores:0')
classes = detection_graph.get_tensor_by_name

('detection_classes:0')
num_detections = detection_graph.get_tensor_by_name

('num_detections:0')
# Actual detection.
(boxes, scores, classes, num_detections) = sess.run(
[boxes, scores, classes, num_detections],
feed_dict={image_tensor: image_np_expanded})
# Visualization of the results of a detection.
vis_util.visualize_boxes_and_labels_on_image_array(
image_np,
np.squeeze(boxes),
np.squeeze(classes).astype(np.int32),
np.squeeze(scores),
category_index,
use_normalized_coordinates=True,
line_thickness=8)
#       plt.figure(figsize=IMAGE_SIZE)
#       plt.imshow(image_np)
cv2.imshow('image',cv2.resize(image_np,(1280,960)))
if cv2.waitKey(25) & 0xFF == ord('q'):
cv2.destroyAllWindows()
cap.release()
break
```

I.2 Arduino Six Axis Robotic Arm Code

```
int delayT = 350;
```

```
void Home() //Call this fucntion when u want to set arm
in home position
{
        MyServo0.write(valGripper);    // Gripper
        delay(15);
        MyServo.write(valBase);    // base
        delay(30);
        MyServo2.write(valShoulder);   // shoulder
        delay(30);
        MyServo3.write(valElbow); // elbow
        delay(delayT);
}

void Pick() // This is fixed pick place.
{
        MyServo.write(4);    // base
        delay(delayT);

        MyServo3.write(125); // elbow
        MyServo2.write(95);   // shoulder
        delay(delayT);

        MyServo0.write(50);    // Gripper open wide
        delay(delayT);

        MyServo0.write(2);    // Gripper close
        delay(delayT);

        MyServo2.write(60);   // shoulder up little
        MyServo3.write(80);   // elbow up little
        delay(delayT);

        Serial.println("Object␣Pickeded");
}
```

```
void Drop() // This is fixed Drop place.
{
        MyServo.write(145);    // base
        delay(delayT);

        MyServo2.write(80);   // shoulder
        delay(15);

        MyServo3.write(115); // elbow
        delay(delayT);

        MyServo0.write(40);    // Gripper open wide
        delay(delayT);

        MyServo3.write(90);   // elbow up little
        delay(delayT);

        Serial.println(F("Object␣Dropped"));
}

void  Playback()
{
        // https://arduino.stackexchange.com/questions

        /1013/how-do-i-split-an-incoming-string
        // input MUST be array ( servoId

        : Position & servoId : Position & servoId : Position )

        //       String phrase;
        //       phrase = String(phrase + ByteReceived);
```

```
 // convert the char input to stirng can be split

// Read each command pair
char* command = strtok(ByteReceived, "&");
while (command != 0)
{
        // Split the command in two values
        char* separator = strchr(command, ':');
        if (separator != 0)
        {
                // Actually split the string in 2:

                replace ':' with 0
                *separator = 0;
                int servoId = atoi(command);
                ++separator;
                int angle = atoi(separator);

                // Do something with servoId and angle

                if (servoId = 1)
                {
                        MyServo.write(angle);
                        delay(delayT);
                }
                else if (servoId = 2)
                {
                        MyServo2.write(angle);
                        delay(delayT);
                }
                else if (servoId = 3)
                {
                        MyServo3.write(angle);
                        delay(delayT);
```

```
                    }

                }
                // Find the next command in input string
                command = strtok(0, "&");
            }

        }
```

APPENDIX II

INVERSE KINEMATIC

II.1 Geometric Approach

Analysing Figure 3.6 and Figure 3.8, the first 3 links of the robot were projected on a Cartesian graph shown in Figure II.1. To calculate θ_1, the coordinate (Px_{tip}, Py_{tip}) and the geometric equation II.1 was used; two values of θ_1 were generated, equation II.2.

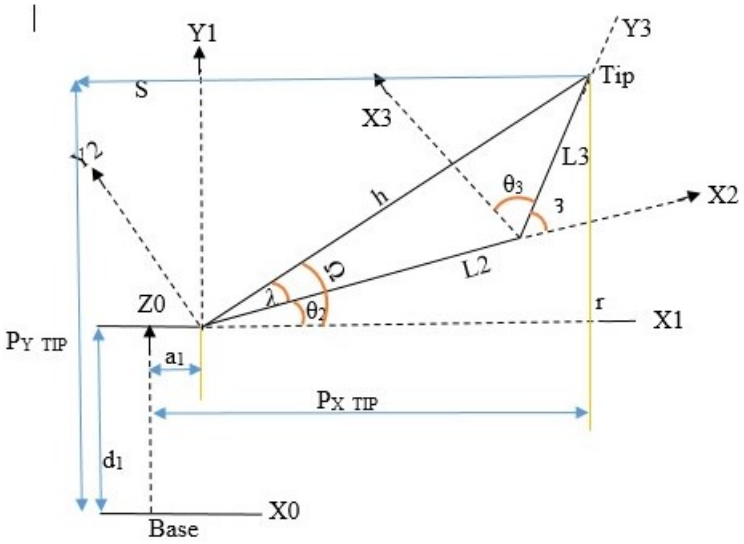


Figure II.1: Projection of links two and three onto the *X-Y* plane.

$$\theta_1 = \arctan2(Py_{tip}, Px_{tip}) \tag{II.1}$$

$$\theta_{11} = \pi + \theta_1 \tag{II.2}$$

θ_3 was next solved using cosine rule as per equation II.3.

$$h^2 = (L_2)^2 + (L_3)^2 - 2.L_2.L_3.\cos(180 - \zeta) \tag{II.3}$$

From Figure 3.8 we notice that:

$$L_3 = d_4 + d_6 \tag{II.4}$$

$$L_2 = a_2 \tag{II.5}$$

$$h^2 = s^2 + r^2 \tag{II.6}$$

$$\cos(180 - \zeta) = -\cos(\zeta) \tag{II.7}$$

Replacing equations II.4, II.5, II.6 and II.7 into II.3.

$$s^2 + r^2 = a_2^2 + (d_4 + d_6)^2 + s.a_2.(d_4 + d_6).\cos(\zeta) \tag{II.8}$$

$$\cos(\zeta) = \frac{(s^2 + r^2 - a_2^2 - (d_4 + d_6)^2)}{2a_2.(d_4 + d_6)} \tag{II.9}$$

Converting the values of s and r in equation II.9 into Px_{tip}, Py_{tip}, Pz_{tip} and θ_1

$$s = Pz_{tip} - d_1 \tag{II.10}$$

$$r = \pm\sqrt{(Px_{tip} - a_1.\cos(\theta_1))^2 + (Py_{tip} - a_1.\sin(\theta_1))^2} \tag{II.11}$$

Substituting equation II.10 and II.11 into II.9.

$$\begin{aligned}\cos(\zeta) =& \frac{(Pz_{tip} - d_1)^2 + (Px_{tip} - a_1.\cos(\theta_1))^2}{2a_2.(d_4 + d_6)} \\ &+ \\ &\frac{(Py_{tip} - a_1.\sin(\theta_1))^2 - a_2^2 - (d_4 + d_6)^2}{2a_2.(d_4 + d_6)}\end{aligned} \tag{II.12}$$

But from geometry identities:

$$\sin(\zeta) = \pm\sqrt{1 - \cos 2(\zeta)} \tag{II.13}$$

$$\zeta = \text{atan2}(\sin(\zeta), \cos(\zeta)) \tag{II.14}$$

Therefore:

$$\theta_3 = -(90 + \zeta) \tag{II.15}$$

The negative sign indicates that the rotation occurred in opposite direction of the Z axis.

The same procedures were followed to get θ_2.

$$\theta_2 = \Omega - \lambda \tag{II.16}$$

$$\Omega = \text{atan2}(s, r) \tag{II.17}$$

$$\lambda = \text{atan2}((d_4 + d_6).\sin(\zeta), (d_4 + d_6).\cos(\zeta)) \tag{II.18}$$

Replacing equation II.17 and II.18 into II.16:

$$\theta_2 = \text{atan2}(s, r) - \text{atan2}((d_4 + d_6).\sin(\zeta), (d_4 + d_6).\cos(\zeta)) \tag{II.19}$$

Replacing the values of s and r from equations II.10 and II.11 into II.19:

$$\begin{aligned} \theta_2 = \text{atan2}(&Pz_{tip} - d_1, \\ &\pm \sqrt{(Px_{tip} - a_1.\cos(\theta_1))^2 + (Py_{tip} - a_1.\sin(\theta_1))^2}) \\ &- \text{atan2}((d_4 + d_6).\sin(\zeta), (d_4 + d_6).\cos(\zeta)) \end{aligned} \tag{II.20}$$

Finally:

$$\theta_2 = -((\Omega - \lambda) - 90) \tag{II.21}$$

There exists multiple solutions for θ_1, θ_2 and θ_3 due to the solutions of the geometric identities meaning several positions can be achieved with many orientations.

II.2 Analytic Approach

After solving the first sub-problem, the next step was to solve the orientation problem and get the remaining angles. This step was solved by using *Z-Y-X* Euler's rotation formula (Kuipers et al., 2009) which is shown in Figure II.2. This formula is used to describe any rotation about a linear independent axes and was used to solve the second part of the 6 dof robotic arm.

The consecutive rotations in Figure II.2 can be explained by the following matrices:

$${}^0_6R = R_{z'y'x'} = R_z(\alpha)R_y(\beta)R_x(\gamma) \tag{II.22}$$

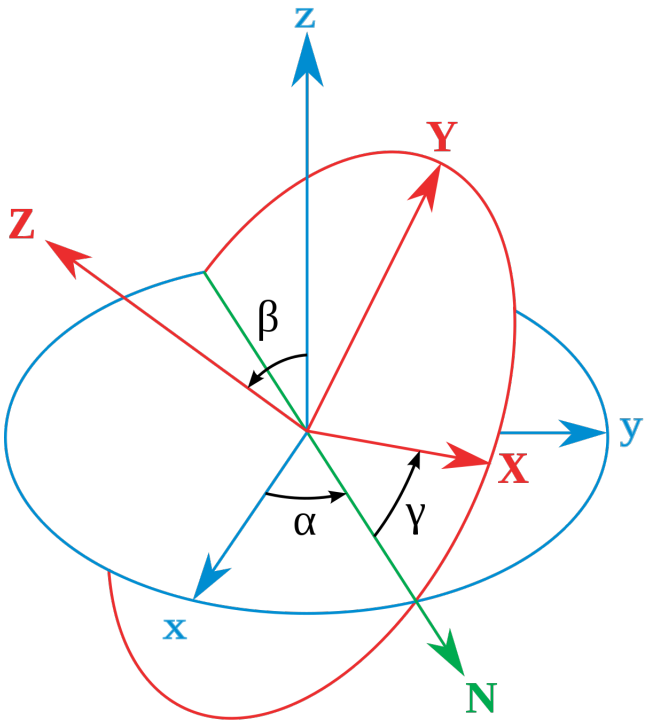


Figure II.2: *Z-Y-X* Euler's rotation method.

$$ {}^{0}_{6}R = R_z(\alpha)R_y(\beta)R_x(\gamma) \tag{II.23} $$

$$ {}^{0}_{6}R = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix} \tag{II.24} $$

$$ {}^{0}_{6}R = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta c\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \tag{II.25} $$

From forward kinematicsKucuk and Bingul (2011):

$$ {}^{0}_{3}R = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & -s_1 \\ s_1c_{23} & -s_1s_{23} & c_1 \\ -s_{23} & -c_{23} & 0 \end{bmatrix} \tag{II.26} $$

$$ {}^{3}_{6}R = ({}^{0}_{3}R)^T({}^{0}_{6}R) \tag{II.27} $$

$$
\begin{aligned}
{}^{3}_{6}R &= \begin{bmatrix} c_1c_{23} & s_1c_{23} & -s_{23} \\ -c_1s_{23} & -s_1s_{23} & -c_{23} \\ -s_1 & c_1 & 0 \end{bmatrix} \\
&\begin{bmatrix} c\alpha c\beta & c\alpha s\beta c\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
\end{aligned} \tag{II.28}
$$

$$
{}^{3}_{6}R = \begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} \tag{II.29}
$$

However, it can be concluded that the last three intersected joints form a set of *ZYZ* Euler angles with respect to frame 3(Shah, Saha, & Dutt, 2009). Therefore, these rotations can be expressed as:

$$
{}^{3}_{6}R = R_z(\alpha)R_y(\beta)R_z(\gamma) \tag{II.30}
$$

$$
{}^{3}_{6}R = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{II.31}
$$

$$
\begin{aligned}
{}^{3}_{6}R &= \begin{bmatrix} c\alpha c\beta c\gamma - s\alpha s\gamma & -c\alpha c\beta s\gamma - s\alpha c\gamma & c\alpha s\beta \\ s\alpha c\beta c\gamma + c\alpha c\gamma & -s\alpha c\beta s\gamma + c\alpha c\gamma & s\alpha s\beta \\ -s\beta c\gamma & s\beta s\gamma & c\beta \end{bmatrix} \\
&= \begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix}
\end{aligned} \tag{II.32}
$$

Now, using Euler's *ZYZ* angle formula, the angles θ_4, θ_5 and θ_6 were determined[]:

$$
\theta_4 = \alpha = \text{atan2}(\frac{g_{32}}{s\beta}, \frac{-g_{31}}{s\beta}) \tag{II.33}
$$

$$
\theta_5 = \beta = \text{atan2}(\pm\sqrt{g_{31}^2 + g_{32}^2}, g_{33}) \tag{II.34}
$$

$$
\theta_6 = \gamma = \text{atan2}(\frac{g_{23}}{s\beta}, \frac{-g_{13}}{s\beta}) \tag{II.35}
$$

It was determined that θ_1, θ_2 and θ_3 had eight possible solutions while θ_4, θ_5 and θ_6 each had two solutions. These values once generated were fed to fed to a script that ensured the angles were between -90 $^\circ$ and + 90 $^\circ$ (180° servo motor limit) so that no servo motor was not overloaded and burnt out.

APPENDIX III

FSR 402 DATA SHEET

The datasheet provided below was used to calculate the amount of force that was needed to grasp every one of the thirteen objects used in the experiment.

INTERLINK ELECTRONICS
Sensor Technologies

FSR 402 Data Sheet

FSR 400 Series Round Force Sensing Resistor

Description

Interlink Electronics FSR™ 400 series is part of the single zone Force Sensing Resistor™ family. Force Sensing Resistors, or FSRs, are robust polymer thick film (PTF) devices that exhibit a decrease in resistance with increase in force applied to the surface of the sensor. This force sensitivity is optimized for use in human touch control of electronic devices such as automotive electronics, medical systems, and in industrial and robotics applications.

The standard 402 sensor is a round sensor 18.28 mm in diameter. Custom sensors can be manufactured in sizes ranging from 5mm to over 600mm. Female connector and short tail versions can also be ordered.

Features and Benefits

- Actuation Force as low as 0.1N and sensitivity range to 10N.
- Easily customizable to a wide range of sizes
- Highly Repeatable Force Reading; As low as 2% of initial reading with repeatable actuation system
- Cost effective
- Ultra thin; 0.45mm
- Robust; up to 10M actuations
- Simple and easy to integrate

Industry Segments

- Game controllers
- Musical instruments
- Medical device controls
- Remote controls
- Navigation Electronics
- Industrial HMI
- Automotive Panels
- Consumer Electronics

Figure 1 - Force Curve

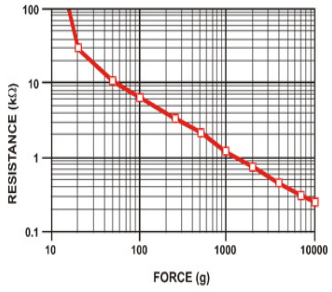
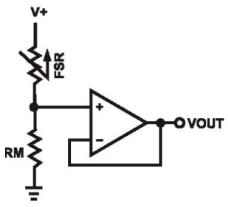


Figure 2 - Schematic



Interlink Electronics - Sensor Technologies