DEEP CONVOLUTIONAL NEURAL NETWORK DESIGN APPROACH FOR 3D-OBJECT DETECTION FOR ROBOTIC GRASPING

by

Purvesh Sharma, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Engineering
December 2020

Committee Members:

Damian Valles, Chair

Heping Chen

Jesus Jimenez

DEDICATION

This thesis is dedicated to my parents and to the people who have supported me throughout my education.

ACKNOWLEDGEMENTS

I would like to thank Dr. Damian Valles, the chairman of my committee as well as my thesis supervisor. As my mentor, with his time, effort, guidance and vision, he has taught me more than I could ever give him credit for. He consistently allowed this paper to be my work but steered me in the right direction whenever he thought I needed it. I would also like to thank him for providing me with the technology required to accomplish this work through HiPE servers.

I would like to thank Dr. Heping Chen for the research opportunity under his guidance. I feel very fortunate for his guidance and support throughout the development of this thesis.

I would like to thank Dr. Jesus Jimenez and Dr. Vishu Viswanathan for guidance throughout my postgraduate degree and for sharing their experiences and knowledge.

The most important part during the span of this thesis was the support and love of my family. I would like to thank my parents; whose love and care gave me strength and who are always supportive of what I pursue.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES viii

LIST OF FIGURES ix

ABSTRACT xiv

CHAPTER

I. INTRODUCTION 1

Contributions 4
Thesis Structure 4

II. BACKGROUND 6

Motivation 6

III. LITERATURE REVIEW 10

RGB-D based 3D-Approach research 10
Amazon Picking Challenge based research 10
Research based on detection techniques 12
Research based on different dataset and classifier comparison 14
Research based on industrial application 14

IV. NEURAL NETWORKS 17

From Biological to Artificial Neuron 19
Biological Neuron 19
Artificial Neural Network 21
Artificial Neuron 22

V. CONVOLUTIONAL NEURAL NETWORK 26

Convolutional Neural Network: an overview 26
Building Blocks of a CNN 27
Convolutional Layers 28

Pooling Layers 31
Activation Layers 34
Fully-Connected Layers 35
Batch Normalization 36
Dropout 37
CNN Architecture 38

VI. METHODOLOGY 40

Dataset Specification 40
Pre-processing 41
Machine Learning Model and Base Classification Network 44
First Designed RGB-D DCNN model 46
Second Designed RGB-D DCNN model 49
Third Designed RGB-D DCNN model 51
Fourth Designed RGB-D DCNN model 53
Training the Designed DCNN model 55
Regularization 56
Hyperparameter Tuning 57

VII. RESULTS AND DISCUSSION 61

Computational Resources 61
Training and Testing Results 63
Performance of Architecture I 64
Performance of Architecture II 68
Performance of Architecture III 78
Performance of Architecture IV 82
For Dark images with architecture II 85

VIII. CONCLUSION 90

IX. FUTURE WORK 92

REFERENCES 93

LIST OF TABLES

Table **Page**

1. Architecture I for RGB-D DCNN model....48
2. Architecture II for RGB-D DCNN model....50
3. Architecture III for RGB-D DCNN model....52
4. Architecture IV for RGB-D DCNN model....54

LIST OF FIGURES

Figure **Page**

1. ABB Robot Arm to be used for hardware implementation 9
2. A Venn diagram describing deep learning as a subfield of neural networks which is a subfield of machine learning all included in AI...18
3. Biological neuron [26] 20
4. Multiple layers in a biological neural network (human cortex) [26] 21
5. An example of 2-layer feed forward artificial neural network 22
6. Artificial Neuron Representation 23
7. Activation functions plot and details [28] 24
8. Relationship between performance and training data for different neural network 25
9. CNN layers with rectangular local receptive fields 28
10. Connection between layers and zero padding..29
11. Reducing dimensionality using a stride of 2 29
12. Convolutional layers with multiple feature maps, and images with three color channels...31
13. An example of a pooling operation with different strides [27]..................................32
14. Max pooling layer with 2x2 pooling kernel, stride two and no padding 32
15. Plots of different activation functions..34
16. An example of an input volume going through a ReLU activation, max (0, x) 35
17. With dropout regularization, at each training iteration a random subset of all neurons in one or more layers except the output layer are dropped out. 37

18. Effect of no Dropout (left) and Dropout with a value of 0.5 (right) 38

19. Typical CNN architecture [26] 39

20. Some Images from dataset after pre-processing in OpenCV are shown 42

21. Image dataset with different objects 43

22. RGB-D DCNN Architecture [1] 44

23. Learning curves for various learning rates (η) 58

24. Relationship between model capacity and loss 59

25. SGD without (on left) and with momentum (on right) 60

26. Features and components of LEAP cluster 62

27. Initial result obtained from architecture I with *lr* 0.0001, for a batch Size of 64, and without pre-processing for 100 x 100 image resolution 65

28. Initial result obtained from architecture I with *lr* 0.0001, for a batch Size of 64, and without pre-processing with 140 epochs for 150 x 150 image resolution 66

29. Accuracy curve with *lr* 0.001, for a batch size of 64 with 80 epochs, and with pre-processing for 150 x 150 image resolution 66

30. Accuracy curve with *lr* 0.001, for a batch size of 128 with 80 epochs, and with pre-processing included in 150 x 150 image resolution 67

31. Accuracy curve with *lr* 0.0001, for a batch size of 32 with 70 epochs, and with preprocessing included in 200 x 200 image resolution 67

32. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 64 with 80 epochs, and with pre-processing included in 150 x 150 image resolution 68

33. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 128 with 80 epochs, and with pre-processing included in 150 x 150 image resolution 69

34. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 32 with 80 epochs, and with pre-processing included in 150 x 150 image resolution 70

35. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 64 with 70 epochs, and with pre-processing included in 200 x 200 image resolution 71

36. Confusion matrix for architecture II for *lr* 0.001, batch size of 64 with 70 epochs, and with the image size of 200 x 200 71

37. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 64 with 80 epochs, and with pre-processing included in 150 x 150 image resolution 72

38. Confusion matrix for architecture II for *lr* 0.001, batch size of 64 with 80 epochs, and with the image size of 150 x 150 72

39. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 32 with 70 epochs, and with pre-processing included in 200 x 200 image resolution 73

40. Confusion matrix for architecture II for *lr* 0.0001, batch size of 32 with 70 epochs, and with the image size of 200 x 200 73

41. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 64 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 74

42. Confusion matrix for architecture II for *lr* 0.0001, batch size of 64, and with 50 epochs with the image size of 200 x 200 74

43. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 64 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 75

44. Confusion matrix for architecture II for *lr* 0.001, batch size of 64 with 50 epochs, and with the image size of 200 x 200 75

45. a) Accuracy b) Loss curve with lr 0.0001, for a batch size of 128 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 76

46. Confusion matrix for architecture II for *lr* 0.0001, batch size of 128 with 50 epochs, and with the image size of 200 x 200 76

47. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 128 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 77

48. Confusion matrix for architecture II for *lr* 0.001, batch size of 128 with 50 epochs, and with the image size of 200 x 200 77

49. a) Accuracy & b) Loss curve with *lr* 0.001, for a batch size of 64 with 80 epochs, and with the image size of 150 x 150 78

50. Confusion matrix for architecture III for *lr* 0.001, batch size of 64 with 80 epochs, and with the image size of 150 x 150 79

51. a) Accuracy & b) Loss curve with *lr* 0.001, for a batch size of 128 with 80 epochs, and with the image size of 150 x 150 80

52. Confusion matrix for architecture III for *lr* 0.001, batch size of 128 with 80 epochs, and with the image size of 150 x 150 80

53. a) Accuracy curve b) Loss curve with *lr* 0.001, for a batch size of 64 with 70 epochs, and with pre-processing included in 200 x 200 image resolution 81

54. Confusion matrix for architecture III for *lr* 0.001, batch size of 64 with 70 epochs, and with the image size of 200 x 200 81

55. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 64 with 70 epochs, and with pre-processing included in 200 x 200 image resolution 82

56. Confusion matrix for architecture IV for *lr* 0.001, batch size of 64 with 70 epochs, and with the image size of 200 x 200 83

57. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 128 with 70 epochs, and with pre-processing included in 200 x 200 image resolution 84

58. Confusion matrix for architecture IV for *lr* 0.001, batch size of 128 with 70 epochs, and with the image size of 200 x 200 84

59. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 128 with 80 epochs, and with pre-processing included in 150 x 150 image resolution 85

60. Confusion matrix for architecture IV for *lr* 0.0001, batch size of 128 with 80 epochs, and with the image size of 150 x 150 85

61. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 128 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 86

62. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 128 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 86

63. a) Accuracy b) Loss curve with *lr* 0.001, for a batch size of 64 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 87

64. a) Accuracy b) Loss curve with *lr* 0.0001, for a batch size of 64 with 50 epochs, and with pre-processing included in 200 x 200 image resolution 87

65. Comparison of time taken by different architectures to train on LEAP server in days 88

66. Confusion matrix for the prediction with the images taken by the camera 89

ABSTRACT

Recognition technology has gained state-of-the-art performance with the dawn of Deep Convolutional Neural Network (DCNN), and with these achievements in computer vision, machine learning, and 3D-sensor, industries are near to start a new era of the automation. However, object detection for robotic grasping in varying environments, low illumination, occlusion, and partial images contributes to poor accuracy and slows the speed of detecting objects. In this thesis, an approach is presented to recognize an object in an industrial warehouse through a robotic vision to advance warehouse automation using a robot arm. A multimodal architecture is designed to be used as a base network/backbone network of Single Shot Detector (SSD) to address the warehouse's challenge, such as partial images and low illumination. This architecture uses Red-Green-Blue (RGB) and Depth images as an input and provides a single output. Most of the researchers used Visual Geometry Group (VGG), Residual Network (ResNet), and MobileNet for detection purposes, but in this thesis, a new DCNN architecture is designed to perform a specific task of grasping. Here, four different Red-Green-Blue-Depth (RGB-D) deep neural network architectures are designed and compared in their training, testing, and other statistical metrics for evaluation. The Deep Convolutional Neural Network (DCNN) development motive is to recognize the input object image obtained from the depth sensor cameras in warehouses. Four objects are mainly focused on this research: '*Bowl*,' '*School glue*,' '*Dove bar*,' '*Soda can*.' Furthermore, details of the designed model, its performance, and the results are discussed in this thesis.

I. INTRODUCTION

With the understanding of 2D-images, understanding of the 3D-images is also promoted from the extraction of features from 3D information in machine learning. Extracting essential features from 3D-images is a perplexing task in computer vision because of its complexity and amount of information. Computer vision applications ranging from mapping, 3D-sensing, robotics, augmented reality, and autonomous driving demand for 3D-feature extraction are in high demand. Many fields in information technology, giant companies, academia, and industries contribute to the 3D-computer vision. The leading cause of the rapid improvement of 3D-vision is its application and necessity in industries [1].

Assessment, image elucidation, and feature extraction of significant features are natural and instantaneous for humans. The robotic vision or computer vision finding the segmentation, orientation, pose, and detection is a profound problem and ambition for the machine learning community [2]. Researchers are trying to imitate human beings' vision ability using high resolution with depth information cameras and convolutional neural networks. In the past decade, CNNs have become a prevalent and powerful tool for the suspension of computer vision problems. However, due to 3D-robot-based vision complexities, computer vision is still in its early infancy [3].

This thesis addresses the design approach in how to improve 3D object detection for a robot arm by utilizing 2D machine learning methods. Here, RGB-D Deep Convolutional Neural Network (DCNN) is designed. The design methods consider the 3D parameters of the images into 2D sets of information and process the object recognition to obtain high accuracy performance of identification and distance information for the navigation and grasp of the robot arm.

In this thesis, a new Deep Convolutional Neural Network (DCNN) architecture for RGB-D images is designed to recognize the warehouse's objects. The research development of the architecture helps a robot arm in recognizing an object on the shelf and tote; so that the robot arm can pick after successfully recognizing it. The model is trained from scratch with the thousands of images for the four classes '*Bowl*,' '*Glue*,' '*Soda can*,' '*Dove bar*' to make this model robust to challenges and complexities of low illumination, different viewing angle, orientation, and partial image. Different viewing angles, orientations, lighting conditions, and brightness were collected and listed as part of the dataset for the training and testing evaluation of the model.

This thesis's deep learning model was selected from four architectures that were evaluated and tested through different metrics. The goal is to design the best architecture which can take Red-Green-Blue (RGB) and Depth images. Both images are fed to the architecture with multiple inputs, RGB and Depth images, and gives a single output as a classification of objects. In this thesis, RGB-D images are used from the dataset, which was taken from the depth sensor in the Massachusetts Institute of Technology (MIT)-Princeton [4] and the University of Washington [5].

The development and implementation of computer vision object recognition to the ASEA Brown Boveri (ABB) company manufactured an industrial robot arm with its controller aims for its gripper movement control. The robot arm is provided with a gripper mounted on it to grasp the objects detected by the robotic vision using the 3D sensor camera. The 3D sensors help obtain a picture in low light and provide depth information of the object to find out the distance between the object and robot arm.

The Deep Convolutional Neural Network (DCNN) method introduced is a multi-modal approach, where a total of four architectures are designed with two different networks for the RGB and Depth network streams in each architecture. These images are fed to the DCNN architecture after image pre-processing using OpenCV. In the DCNN architecture, features are extracted using convolutional layers and pooling layers to sub-sample the images. The output of these two layers is merged and fed to another convolutional layer for classification purposes. Experimentation work is performed under different illumination and challenging conditions such as low light and partial images with the different learning rates, batch sizes, image resolutions, and calibration of the hyperparameters in the architecture.

Contributions

The main contributions of this thesis can be summarized as follows:

- The designed architecture tested on the renowned RGB-D dataset, and it outperforms the current state of the art on the RGB-D Object dataset in [6].
- Object recognition using deep learning is applied to RGB-D images of objects in different poses, varying illumination in a challenging real-world and noisy environment using the data augmentation techniques on RGB-D Washington and Princeton dataset while maintaining good accuracy.
- A new RGB-D DCNN architecture is designed, tested, and compared with the other designed architectures for accurate classification of objects.

Thesis Structure

The thesis is structured into nine main chapters. In this chapter, a brief introduction was given with the problem statement and contribution listed. In the coming Chapter 2, named “Background,” discusses the motivations behind this research work is reviewed. The subsequent Chapter 3 conveys the relevant literature review based on the work done in object detection in research, industries, and challenge competitions. Chapters 4 and 5 discuss the basic principle of the neural networks and the CNN modeling techniques' fundamental concepts.

Chapter 6, named "Methodology," delivers the RGB-D dataset's details for this research, the various preprocessing steps, and the object recognition multi-modal architecture design approach. Chapter 7, entitled "Results and Discussion," presents experiment results and discusses the performance obtained by designed multi-modal architecture. This research work is concluded in chapter 8 with the final observations and summary of the research work. Finally, in Chapter 9, some implications for further research are described.

II. BACKGROUND

This chapter presents the motivation and factors to be considered for this thesis project. It also reviews the Amazon Picking Challenge, robot arm with its controller, and discusses the necessity of computer vision object detection solution for warehouse problems to accomplish their tasks efficiently and effectively.

Motivation

Robots, or cobots (collaborative robots) or assistive robots, have been instrumental in performing tasks that are unfeasible for operators to attain. Robots raise efficiency and safety levels due to their potential of intelligent software integration, dexterity, adaptability, and customization of movements for different operating tasks. Situations like the COVID-19 pandemic, which led to a partial or complete shutdown of plants and factories, resulted in economic downturn that the focus has turned to robotic automation with computer vision solutions . For this, manufacturing companies may need to reconfigure their workplace, warehouses, and factories by replacing operators with intelligent machines as a necessity in terms of economics and supply chain continuation. The practice of social distancing may also hasten the process of automation.

After the commencement of the Amazon Picking Challenge (APC) in 2015, the idea of object detection and recognition attracted attention for the pick and place process of an object from warehouse shelves and place it to a tote. The APC competition challenged the robotics research community to integrate the state-of-the-art in objection and perception, path planning, motion planning, and clasp planning for a complete automation so that robots can perform a task independently. They comprised a streamlined model of the task to pick items from the shelves and put them into receptacles, which is generally faced by operators in warehouses [1].

This thesis focuses on the identification of objects for warehouse and industrial applications in where operators need to manage several objects. The items concentrated for this work are a '*Bowl,*' '*Soda can,*' '*School glue,*' and a '*Dove bar.*' Also, situations like low illumination, partial view of objects, at times is difficult and lethargic for operators, and can be effortless for robots with smarter integration of software and detection models. However, notwithstanding remarkable performances, the problem for robots to detect things accurately and efficiently still challenges scientifically when dealing with real scenarios. The Deep Convolutional Neural Network (DCNN) is designed to classify the objects to overcome the problem of warehouses where there are amorphous objects, different shapes, and structure with blur, noisy images, and complex shapes, which makes it difficult to identify the object. Machine learning tools like TensorFlow, Keras are used in the Python programming language for the design development of the DCNN model, and the training, validation, and testing of the model are performed in the LEAP cluster and a HiPE server.

This thesis provides a vision system design to industrial robot arm capabilities to identify objects for classification. The robot arm and its controller from the ASEA Brown Boveri (ABB) company is shown in Figure 1. It is the fourth generation of robotic technology with six degrees of freedom (DoF). It is an adaptable solution for object handling and assembly applications in electronic, manufacturing, and the food industry. It can handle a load of 3kg. and enabled with fast acceleration with high accuracy for many applications. The movements are executed through the IRC5 (Industrial Robot

Controller) robot controller to control the arm’s rotation, extension, and grasping. It provides the ability to perform a task in an efficient manner providing a pinpoint accuracy path. The IRC5 is programmable with Rapid, a high-level programming language for the ABB robot, which can be integrated with Python. An implementation of a vision system for detection and recognition of 3D-objects in Red-Green-Blue-Depth (RGB-D) images are captured by the 3D sensor.

The Deep Convolutional Neural Network (DCNN) designed in this thesis is a multi-modal approach that can be used as a base or backbone network for object detection network. In this designed network, two inputs are given, RGB and Depth image, with the image processing added in it. Image preprocessing is an indispensable part of this process to detect an object in the low illumination, cluttered background, and partial images. In this approach, a parallel architecture is designed to handle the RGB and Depth image input from the 3D sensor. These two streams of RGB and Depth network are concatenated to combine information from both of the streams. After combining these two streams, an output layer is used for the multi-class classification of objects.

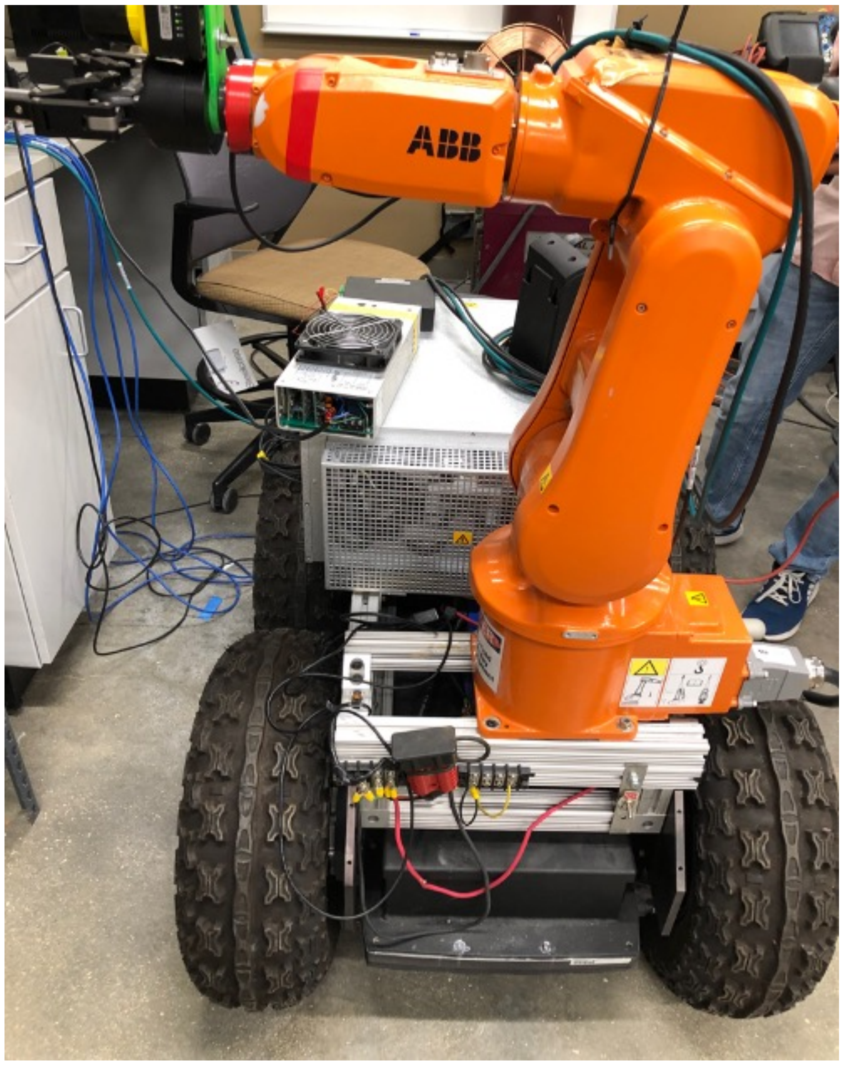


Figure. 1 ABB Robot Arm to be used for hardware implementation.

III. LITERATURE REVIEW

The literature review covers papers based on object recognition, detection in the industries warehouse, and robotic arm movements. Most of the authors discussed deep learning methods in warehouse automation in a semi-structured environment such as low illumination, cluttered objects, and occluded objects.

RGB-D based 3D-Approach research

Marcelo Borghetti Soares et al. [7] explained the problem of robotic vision and processing of RGB-D images, mixing volumetric, and multi-model view representation. The authors evaluated their proposed Point Cloud Object Recognition model using the Washington dataset of RGB-D images and implemented two independent Convolutional Neural Networks (CNN) for learning 3D features. It took 0.0896 seconds for classifying the object using their approach. The accuracy achieved by the designed model was 81.29% ± 9.25 with RGB+D and 86.82± 3.83 with RGB + RGB-D.

Sulabh Kumra et al. [3] evaluated the RGB-D images from Cornell Dataset for robotic grasp detection and challenges related to robotic grasping. The authors mentioned and experimented with two models, Uni-Model and Multi-Model grasp detector, using the ResNet50 model and evaluated the performance of object recognition and grasping. The authors found 88.53% and 94.3% for grasp prediction with these models.

Amazon Picking Challenge based research

Rico Jonschkowski et al. [8] performed in Amazon Picking Challenge (APC) in 2015 and won with their probabilistic multi-class segmentation from RGB-D data. The APC challenge goal is to design an autonomous pick-and-place robot to increase the efficiency and accuracy with precision in the warehouse environment. Here, researchers computed pixel-wise probabilities for different objects and environments, such as reflective, translucent, cluttered, or loosely coupled components. The development used the extended version of the Histogram back projection to a probabilistic version with additional non-color features and also effectively utilized iterative closet point (ICP) by the random forest classifier. Finally, the proposed developed model achieved 83.33% accuracy.

Andy Zeng et al. [9] presented about what they achieved in the APC 2016. The authors designed a vision system to perform smoothly under cluttered objects, self-occlusions, and a large variety of objects using a multi-view Self-supervised Deep Learning algorithm. The model used 6-degrees of freedom ABB IRB1600ID robot industrial manipulator with the RealSense F200 camera on it. The model was evaluated with performance metrics separately for cluttered, occlude object, and vision prediction accuracy ranges from 60% to 80%.

A.P del Pobil et al. [10] participated in the APC to automate the pick-and-place operations in an unstructured environment. The authors used two 7-degrees of freedom robotic arms for object recognition. For feature matching, SIFT (Scale Invariant Feature Transform) to identify objects and approximate the nearest neighbor matching to improve query time is used. To increase performance and accuracy, SegNet and ResNet are also used, which are CNN based. The three algorithms were integrated and achieved an improved performance per item and reduced timing by eleven seconds.

Carlos Hernandez Corbato et al. [11] focused their work mainly on product handling using a robotic arm, which involves pick and place in a warehouse's semi-structured environment. The authors adopted the Faster R-CNN method and visual geometry group network (VGGNet) as the initialization for the convolution filters and gaussian distribution for object detection. With this deep learning algorithm, the Delft Robotic team secured the first position in the APC 2016 with a mean pick-and-place time of 33.76 seconds and an efficiency of 97.7%. The authors evaluated their CNNs and found the mean average precision for Bin and Tote model as 82.9% and 90.0%.

Max Schwarz et al. [12] also presented in APC 2016 to overcome the problem of handling a large variety of items in the warehouse. The authors achieved a success rate of 83.33% (picked 10/12 objects) using a semantic segmentation and SVM object detection approach. Both approaches are based on CNN pre-trained on ImageNet dataset images.

Colin Rennie et al. [13] proposed a method to detect objects using RGB-D (Red, Green, Blue, and depth) sensor for warehouse pick-and-place robot with different challenging conditions of low illumination inside shelves, texture-less, and reflective objects. This idea came from the first APC competition. The author proposed a new dataset to increase the accuracy of object detection and pose estimation in the warehouse, and there are always complications of smaller bins, low illumination, multiple objects, clutter, similar objects, and overcoming these obstacles.

Research based on detection techniques

Xin Lu et al. [14] replaced the base VGG16 network with ResNet101, keeping the SSD Network the same as the original to detect dangerous goods, such as knives, revolvers, and rifles. The replacement of architecture helped improve accuracy by 17.40%, which was 55.0% with VGG16, but the amount of calculation increased with these changes.

Alexander Womg et al. [15] designed an SSD CNN named Tiny-SSD for real-time object detection to handle memory issue requirements and computational cost. It is 26X times smaller than Tiny-You Only Look Once (YOLO) with a size of 2.3MB and achieved a mean Average Precision (mAP) of 61.3%, which is ~4.2% higher than Tiny-YOLO on the Visual Object Classes (VOC) 2007 dataset.

Shen-Fu Hsiao et al. [16] designed a model to reduce the VGG DNN model's complexity without sacrificing accuracy in object classification and detection. The authors named the Low Complexity DNN and achieved an accuracy of 86.3% in classification and 70% in detection on the ImageNet dataset.

Chengcheng Ning et al. [17] proposed an improved SSD algorithm to increase accuracy without affecting speed. The authors implemented an Inception Block to the SSD with some extra Batch Normalization (BN) layers. The main idea of Inception was to use dense components to approximate the optimal local sparse structure. The authors believed that the Inception block could catch more information without increasing network complexity. This implementation achieved an accuracy of 78.6% on the Pascal VOC2007 dataset.

Wei Liu et al. [18] presented a detecting object in an image using deep neural networks. This was the first SSD model implemented in December 2016 for object detection. It achieved an accuracy of 76.9% mAP and outperformed the Faster R-CNN model on Pascal VOC 2007 dataset, which includes tv, bottle, chair, bus, car, and other object classes.

Zhu Dongtao et al. [19] focused on solving traffic sign detection's common precision problem by improving the SSD detection algorithm. In this research, the authors removed some high-layer features maps, adjusted the low-layer feature maps, and adjusted the aspect ratio. These changes achieved mean Average Precision of 75.28% under different environmental conditions and outperformed classical SSD by ~10%

Research based on different dataset and classifier comparison

Arul Selvam et al. [20] shed light on warehouses' challenges, such as adding new objects continuously and frequent human interaction. To overcome the situation, the authors proposed a pipeline to minimize human intervention and increase the reliability of the 3D-models of objects. Authors trained a CNN for 6D-pose estimation from RGB images of a single class and multiclass images in a cluttered scene and maximum occlusion, limited to 50%, including symmetric and asymmetric objects. The method is evaluated on a synthetic validation dataset and cluttered in a real-world scene.

Hyun-Jun Jo et al. [21] Synthesize dataset consisting of 10,000 images of objects, including different backgrounds, and implemented Faster R-CNN based on the VGG16 Net model for object recognition for bin picking system. The authors explained that deep learning algorithms need to be fine-tuned to improve the efficiency and accuracy of grasping objects. In their proposed method, four main steps are considered: the generation of the dataset using the dataset synthesis method, train the object recognition algorithm by the generated dataset, calculate the center points of objects, and grasp objects. The authors' approach achieved a 99% success rate (149/150).

Research based on industrial application

Kyekyung Kim et al. [22] proposed multiple object recognition solutions for the manufacturing process in industrial robot applications for diverse objects, including reflective surface and complex shape, illumination effect. The authors used 2D- and 3D-vision sensors for occluded objects, and the difference of Gaussian filters are used to overcome illumination effects. A three-layer perceptron model is implemented and obtained an accuracy of 96% to classify objects.

Rasmus S. Andersen et al. [23] presented the scenario of industries where traditional robots are used to set up and perform a simple, repetitive pick and place task. The authors proposed object recognition for pick-and-place using SVM, Logistic Regression, k-Nearest Neighbor (kNN), and ANN classifiers. The object chosen belongs to a challenging group of objects with a highly reflective surface, metallic, and colorless in a semi-structured industrial environment. The authors found the highest accuracy with a Logistic regression classifier of 93% and slightly better than SVM in experimentation.

Kyekyung Kim et al. [24] explained different issues and challenging conditions in manufacturing plants for pick-and-place robots. Precision is required for assembly and large-scale assembly automation that utilizes computer vision navigating different lighting conditions and environments. The authors proposed a dynamic object recognition using precise location detection, which is not restricted to shape, distance, location, and illumination conditions using a neural network. It includes randomly located multiple types of objects, and visual serving object randomly piled bin picking targeted objects. Their work found a 0% recognition error for training object samples and 2.2% error for testing object samples.

Xi Chen et al. [25] performed pick and place experiments on industrial tools and office supplies using Universal Robot 5 (UR5). The authors used 50 different classes, including tools and office supplies, to increase variation samples in the dataset and rotated all samples to different angles. Faster R-CNN based on VGG-M net was applied to the vision system for object detection. The authors tested the system using two cameras, Kinect and ZED, and the ZED camera was chosen over Kinect due to its signal transmission ability. The work found the object detection accuracy of 68% using Faster R-CNN.

The literature survey discusses SSD techniques, improvement in base network, dataset, different image preprocessing tools, classifiers, and camera types. The discussed papers provide different ideas and implementation knowledge about the problems faced with object detection in different environments, industries, different algorithms, and datasets. In this thesis, an architecture is designed that addresses all the problems discussed to provide a specific classification of objects in simulated warehouse environmental conditions.

IV. NEURAL NETWORKS

Machine learning is a subset of Artificial Intelligence (AI) responsible for providing systems the ability to automatically learn and evolve from experience from data and observations available without being explicitly programmed. Here is a slightly more general definition of machine learning:

"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed." Arthur Samuel, 1959

, and a more engineering-oriented definition of machine learning:

"A computer program is said to learn from experience ***E*** *with respect to some task* ***T*** *and some performance measure* ***P****, if its performance on* ***T****, as measured by* ***P****, improves with experience* ***E.****"* Tom Mitchell, 1997

Deep learning is a subset of Artificial Neural Network (ANN), and ANN is a subset of machine learning. A graphical depiction of the relationship between AI, machine learning, neural networks, and deep learning is shown in Figure 2.

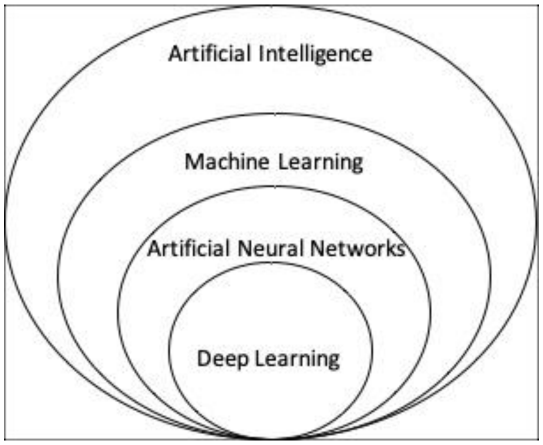


Figure 2. A Venn diagram describing deep learning as a subfield of neural networks which is a subfield of machine learning all included in AI

Many human achievements in this society are inspired by nature, like electricity, planes, bullet trains, Sonar, Velcro, and nature has inspired countless more inventions. Similarly, the idea of ANNs is also inspired by the networks of biological neurons found in our brains. ANNs are at the very core of deep learning; they are versatile, robust, and scalable. It makes them ideal for tackling large and highly complex machine learning tasks such as classifying billions of images, powering speech recognition services, and recommending the best videos to watch hundreds of millions of users every day [26].

Training computers to learn like humans' brains is achieved partly through ANN. ANNs are a series of parametric algorithms whose design is inspired by the structure of biological neural networks that constitute the human brain. The model capability to handle more complex classification tasks increases, and it is referred to as a deep network as the number of hidden layers increases. Deep learning belongs to the family of ANN algorithms. However, deep learning networks are distinguished from the more commonplace single hidden-layer neural networks by their depth; that is, the number of node layers through that data must pass in a multistep process of pattern recognition. There is no consensus amongst experts on the depth of a network to be considered.

In the next sections, the building blocks of neural networks and the algorithms used to make the network able to learn would be discussed. Furthermore, techniques to speed up the learning process and enhance the deep learning model's performance will be reviewed.

From Biological to Artificial Neurons

Surprisingly, ANNs were first introduced in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts in [26]. McCulloch and Pitts presented a simplified computational model of how biological neurons might work together in animal brains to perform complex computations using *propositional logic*. This was the first artificial neural network architecture conceptual approach in the forms of definition and design. Since then, many other architectures have been developed and innovated for various applications and types of data formats.

Biological Neurons

Before discussing artificial neurons, the biological neuron is represented in Figure 3. It is an unusual-looking cell mostly found in animal brains. It comprises a *cell body* containing the nucleus and most of the cell's complex components, many branching extensions called *dendrites*, and one long extension called the *axon*. The axon's length maybe just a few times longer than the cell body, or up to tens of thousands of times longer. Near its extremity, the axon splits off into many branches called *telodendria*, and at the tip of these branches are minuscule structures called *synaptic terminals* (or simply *synapses*), which are connected to the *dendrites* or cell bodies of other neurons. Biological neurons produce short electrical impulses called *action potentials* (APs, or just *signals*) that travel along the axons and make the synapses release chemical signals called *neurotransmitters*. When a neuron receives a sufficient amount of these neurotransmitters within a few milliseconds, it fires its electrical impulses due to the *neurotransmitters* where some of them inhibit the neuron from firing [26].

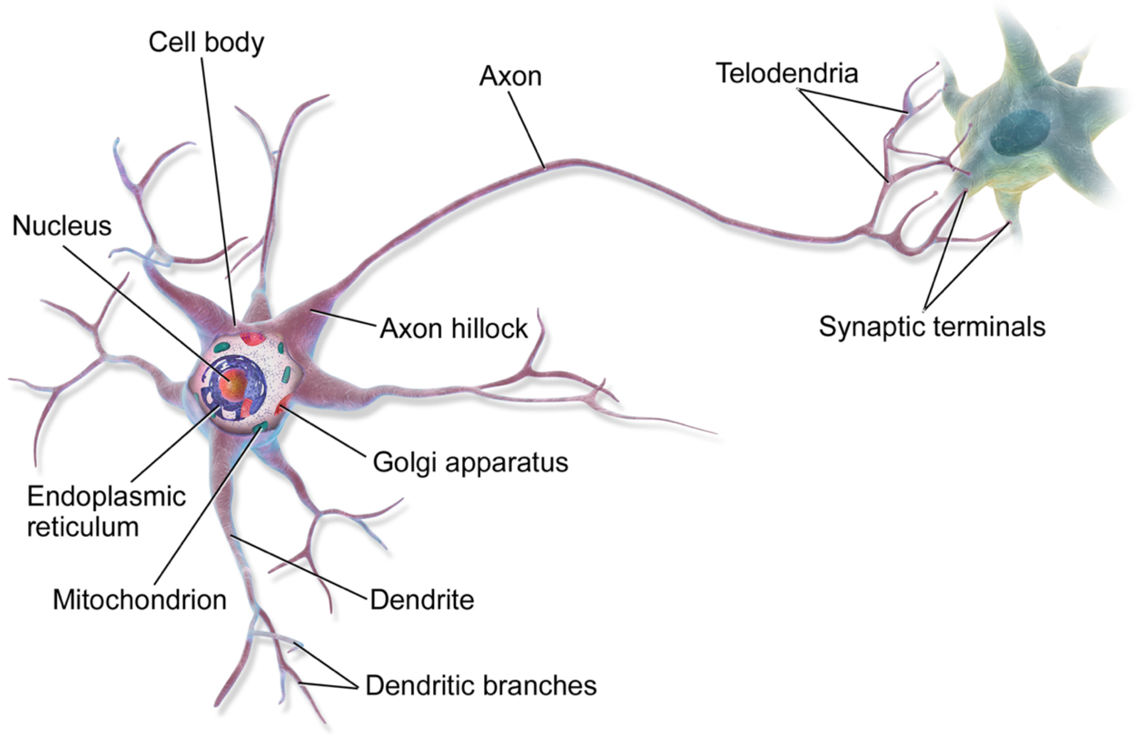


Figure 3. Biological neuron [26]

Thus, individual biological neurons seem to behave in a relatively simple way, but they are organized in a vast network of billions, with each neuron typically connected to thousands of other neurons. Highly complex computations can be performed by a network of reasonably simple neurons, much like a complex anthill can emerge from simple ants' combined efforts. The architecture of biological neural networks (BNNs) [26] is still the subject of active research, but some parts of the brain have been mapped, and it seems that neurons are often organized in consecutive layers, especially in the cerebral cortex, the outer layer of the brain, as shown in Figure 4.

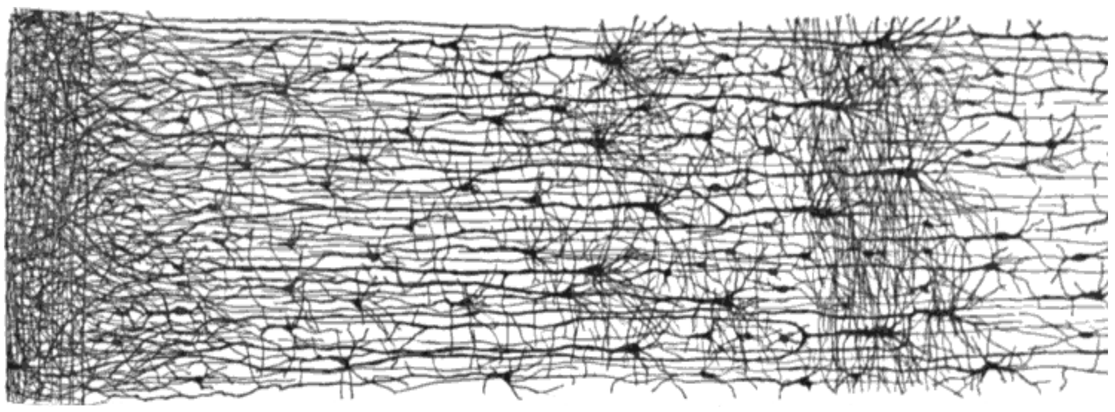


Figure 4. Multiple layers in a biological neural network (human cortex) [26]

Artificial Neural Network

ANN is formed by artificial neurons, where those neurons are organized in layers. There are three types of layers: an input layer, one or multiple hidden layers, and a single output layer. Neural networks are classified from their number of hidden layers and how they are connected. If the neural network has/or not loops can be classified as Recurrent or Feed-forward neural networks. An example of a two-layer feed-forward artificial neural network is shown below in Figure 5. Imagine that the connections between neurons are the parameters that will be learned during training. In this example, Layer L1 is the input layer, L2 and L3 are the hidden layers, and L4 is the output layer.

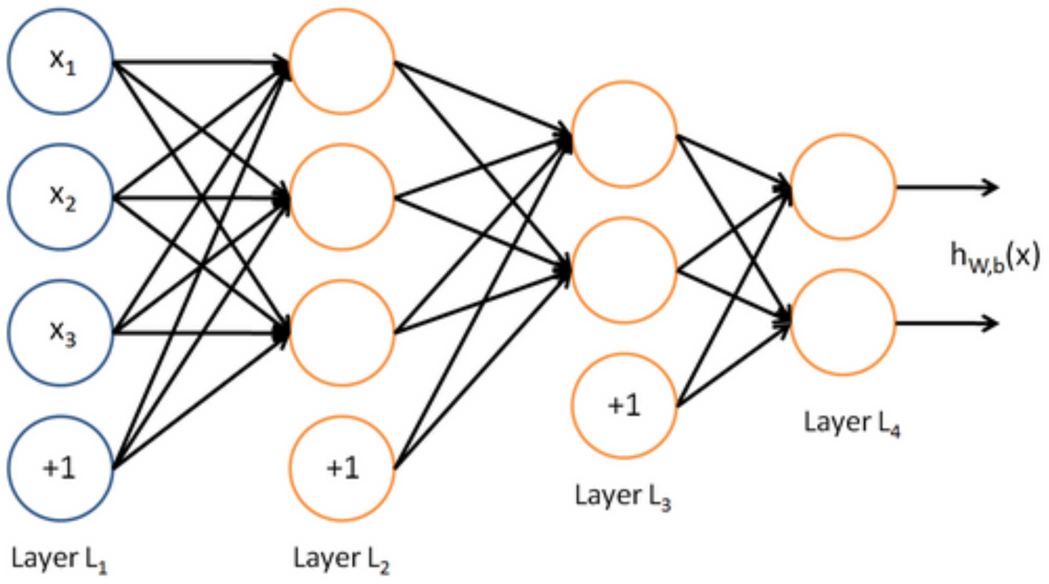


Figure. 5 An example of 2-layer feed forward artificial neural network

Artificial Neuron

The single artificial neuron shown in Figure 6 will perform a dot product between *w* and *x*, including the *bias +1* node, and the result is passed to an activation function that will add some non-linearity. This non-linear function is called the activation function. Different types and more details of the activation functions is shown in Figure 7. In the past, the popular choice for activation functions was the sigmoid and tanh [26][27]. However, when *n* number of hidden layers use an activation like the sigmoid function, *n* small derivatives are multiplied together. Thus, the gradient decreases exponentially as it propagates down to the initial layers. The Rectified Linear Unit (ReLU) layers is an easy solution, as it does not cause a small derivative, it converges faster, and the slope does not saturate when it gets large to overcome this problem. Hence, a better response for deep neural networks. Therefore, ReLU neurons are considered in the research.

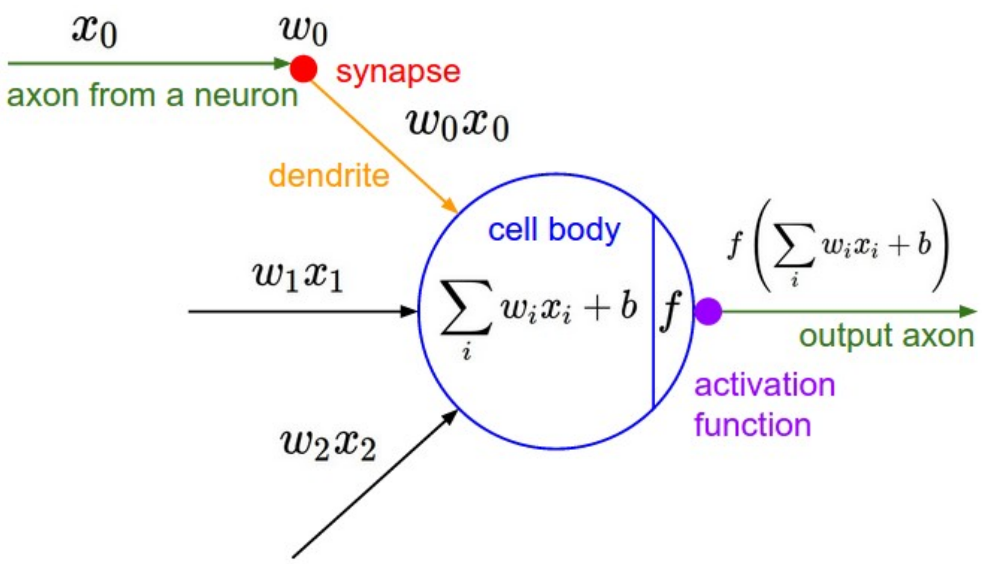
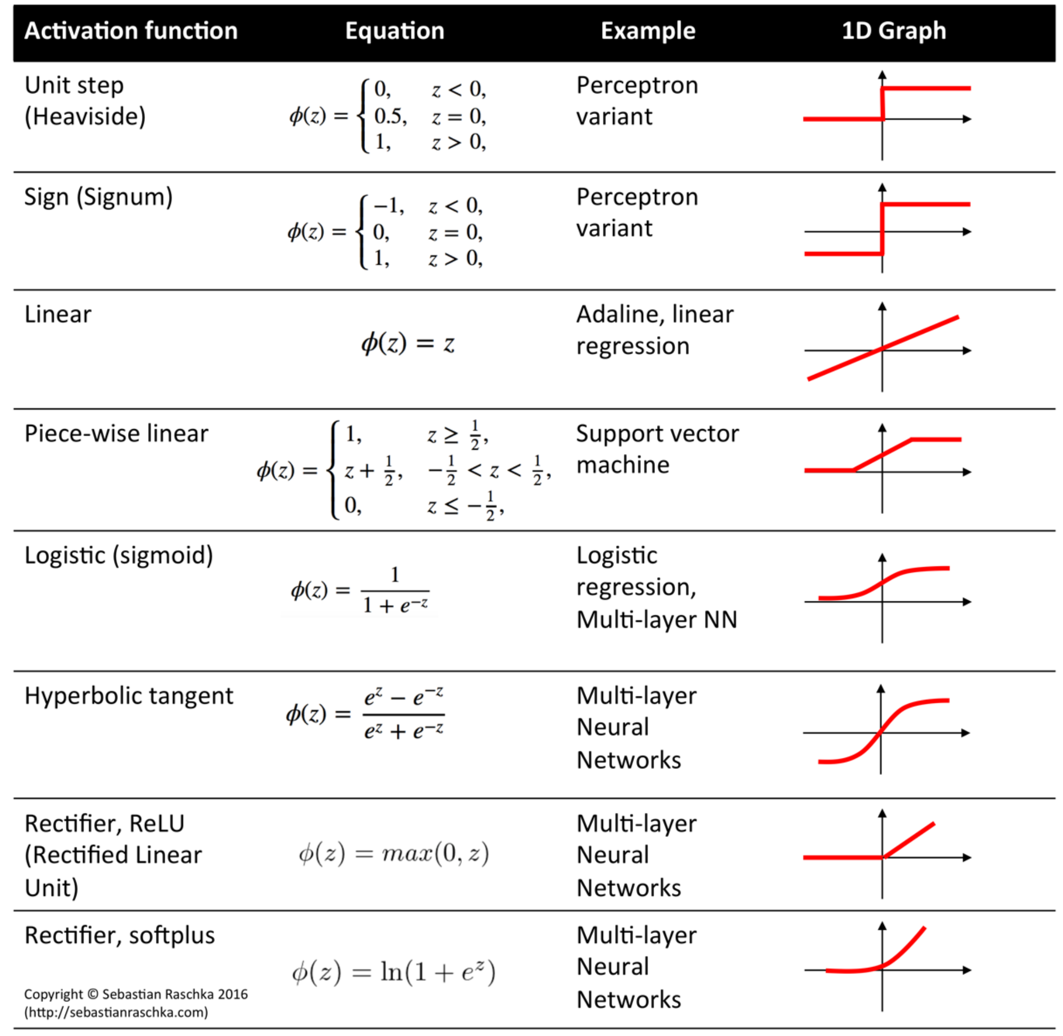


Figure 6 Artificial Neuron Representation

Those artificial neurons will form the neural network. The non-linearity will allow different variations of the same class object to be learned separately, which is different behavior than the linear classifier that tries to learn all different variations of the same class on a single set of weights. When an ANN contains a deep stack of hidden layers, it is called a deep neural network (DNN). The field of deep learning studies DNNs, and more generally, models containing deep stacks of computations. Many practitioners discuss deep learning whenever neural networks are involved, including shallow ones. More neurons and more layers are always better, but it will need more data to train. Each layer learns a concept from its previous layer. Therefore, it is beneficial to have deeper neural networks than a wide one because the layer does not need to learn the whole concept at once. Instead, the deep neural network builds a chain of features that constructs the dissection of information weighted to a proper decision or outcome. As the depth of DNN increases, the classification accuracy also increases opposed to the case for traditional machine learning algorithms.



Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Figure 7 Activation functions plot and details [28]

As per the depth of the network, the behavior of neural networks can be better shown by a plot inspired by Andrew Ng's 2015 talk in [27] in Figure 7. From the plot, it can be seen that as the amount of training data increases, accuracy also increases for DNNs, whereas it comes to a standstill for traditional machine learning algorithms. This is the reason to associate large datasets with deep learning. For image classification, if the dataset contains more number of images, then DL may achieve better results compared to other machine learning algorithms such as logistic regression, SVMs, decision trees, and other techniques [27].

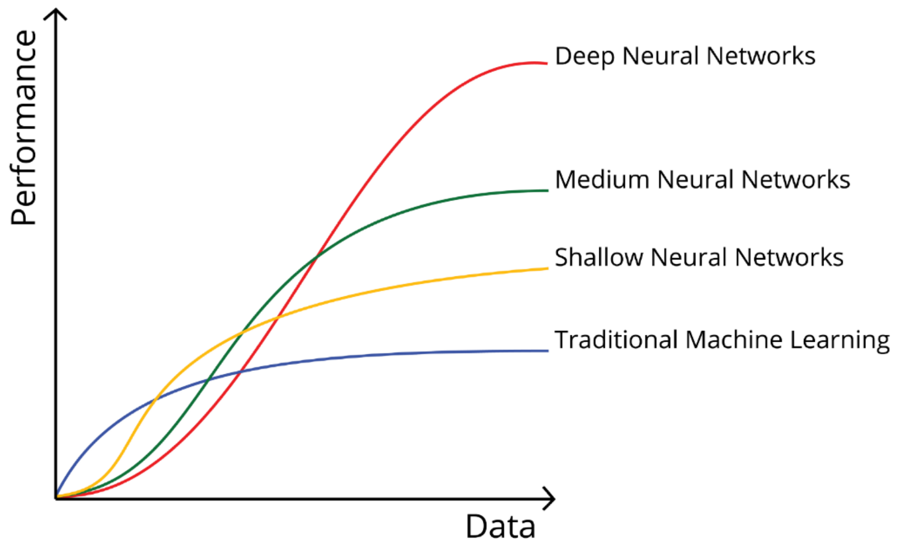


Figure 8. Relationship between performance and training data for different neural networks

In this chapter, the fundamentals of Deep Learning are discussed, which belongs to the family of Artificial Neural Networks (ANNs). In the next chapter, Convolutional Neural Network is discussed, it is a subset of Deep Neural Network.

V. CONVOLUTIONAL NEURAL NETWORK

The chapter describes the Convolutional Neural Network (CNN) learning model in the context of object recognition. A detailed introduction to basic concepts of convolution and basic building blocks of the CNN is also discussed.

Convolutional Neural Network: an overview

Convolutional Neural Network (CNN) belongs to a powerful family of ANNs. It is specially designed to handle computer vision problems. The CNN emerged from the brain's visual cortex study and used in image recognition since the 1980s. In the last few years, CNNs have accomplished herculean performance on some complex visual tasks due to the amount of available training data and increased computational power. CNNs are the backbone to power self-driving cars, image search services, automatic video classification systems, and an increase of image and video applications.

In the last decade, the number of architectures has been developed and tested on different datasets like ImageNet, Pascal VOC. Traditional feed-forward machine learning algorithms use Fully Connected layers with each neuron of one layer is connected to every output of the previous layer. A CNN starts with a convolutional layer and is connected to a Fully-Connected layer at the end of the architecture, making it very efficient because convolution layers carry fewer parameters than Fully-Connected layers reducing the memory requirements and computational cost. Each layer of a CNN applies a different set of filters and combines the results by feeding the output into the network's next layer. During the training of a model, CNN automatically learns the values for these filters by pixel. “In the context of image classification, CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., “blobs”) in the second layer.
- Use these shapes to detect higher-level features such as facial structures, parts of a car, or any features in an image in the highest layers of the network.” [27]

Building Blocks of a CNN

There are different types of layers in a CNN network. The most common layers of a CNN are as follows:

- Convolutional Layer
- Activation Layer
- Pooling Layer
- Fully-Connected Layer
- Batch Normalization Layer
- Dropout Layer

Convolutional Layers

The essential building block of a CNN architecture is the convolutional layer. Neurons in the first convolutional layer are not connected to every pixel in the input image, but only to pixels in their receptive fields. Each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer shown in Figure 9. This architecture allows the network to concentrate on small low-level features in the first hidden layer, then assemble them into more extensive higher-level features in the next hidden layer, and so on. This hierarchical structure is standard in real-world images, which is one reason why CNNs work so well for image recognition.

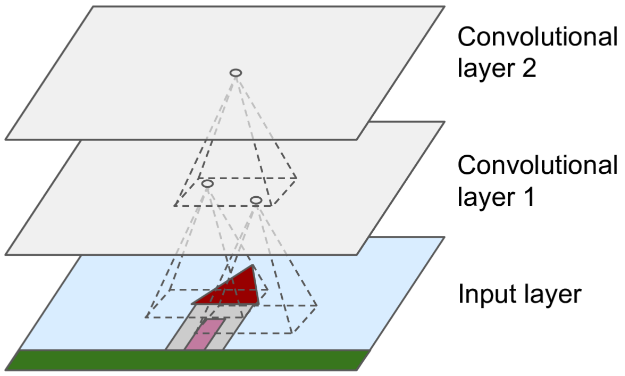


Figure 9. CNN layers with rectangular local receptive fields

In a CNN, each layer is represented in 2D, making it easier to match neurons with their corresponding inputs. It consists of a set of learnable filters or kernels. These kernels are usually of small sizes, such as 1x1 or 3x3. These filters slide across the input image; a filter/kernel is also an array of numbers called weights or parameters. A significant note is that the depth of this filter has to match the depth of the input. It is common to add zeros around the inputs, as shown in Figure 10, for a layer to have the same height and width as the previous layer. The addition of zeros is known as zero padding.

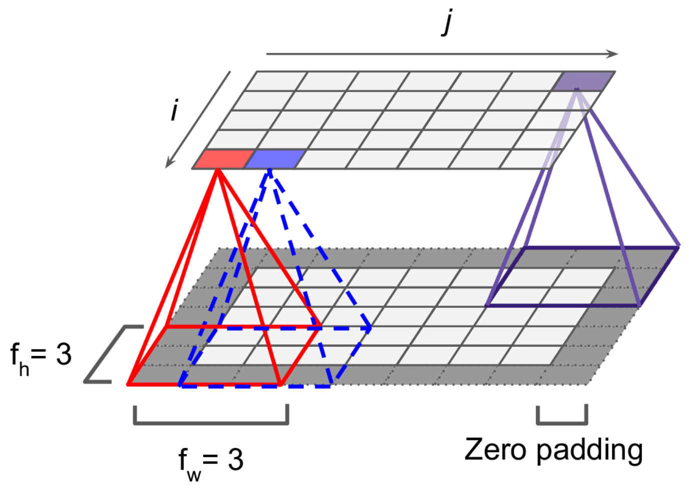


Figure 10. Connections between layers and zero padding

It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields, as shown in Figure 11. It will dramatically reduce the model's computational complexity by skipping the intermediate location to reduce the image's resolution. The shift from one receptive field to the next is called the stride.

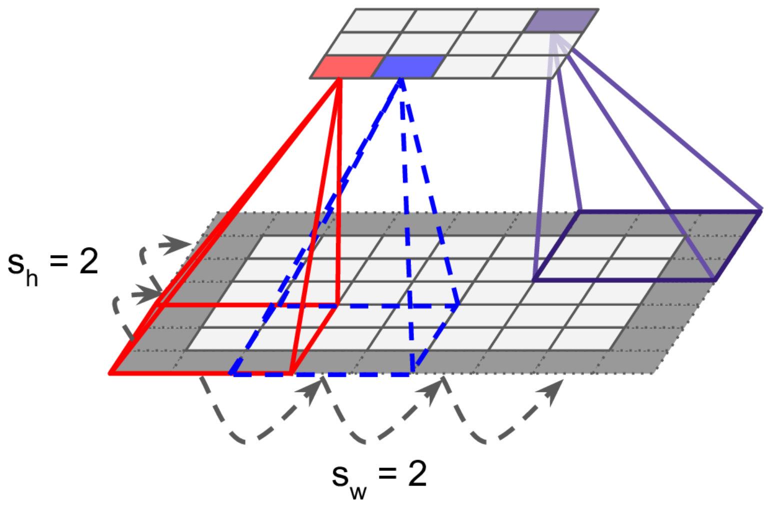


Figure 11. Reducing dimensionality using a stride of 2

The following formula can be used to find the size of the first two dimensions of the output volume:

$$(W - F + 2P)/S + 1 \qquad \text{Eq. 1}$$

, where *W* is the input volume size, *F* represents the filter size of the convolutional layer, *S* and *P* are the stride and amount of zero padding, respectively. In Figure 3, a 5×7 input layer plus zero padding is connected to a 3×4 layer, using a 3×3 receptive fields and a stride of two.

- padding must be either "Same" or "Valid":
    - If set to "SAME," the convolutional layer uses zero padding if necessary. The output size is set to the number of input neurons divided by the stride, rounded up. For example, if the input size is 50 and the stride is 3, then the output size is 17 (i.e., 50/ 3 = 16.6, rounded up to 17). Then zeros are added as evenly as possible around the inputs, as needed. When strides=1, the layer's outputs will have the same spatial dimensions (width and height) as its inputs, hence the name same.
    - If set to "VALID," the convolutional layer does not use zero padding and may ignore some rows and columns at the bottom and right of the input image, depending on the stride. This means that every neuron's receptive field lies strictly within correct positions inside the input, hence the name valid.

Input images are also composed of multiple sublayers: one per color channel. There are typically three channels to generate a colored image with red, green, and blue (RGB) shown in Figure 12. Grayscale images have just one channel, but some images may have much more—for example, satellite images that capture extra light frequencies, such as infrared and hyperspectral images.

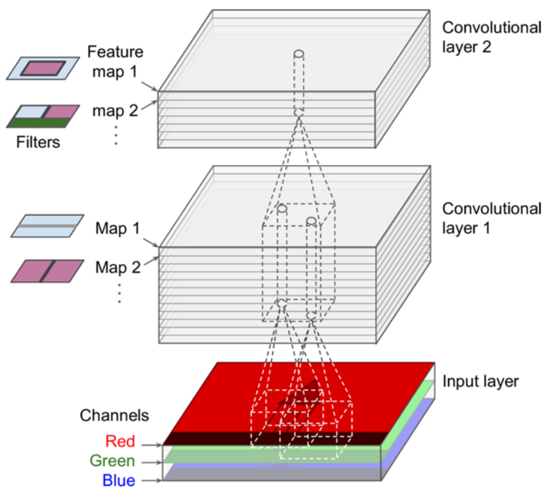


Figure 12. Convolutional layers with multiple feature maps, and images with three color channels

Pooling Layers

Pooling layers are used to subsample the input image to reduce the computational load, memory usage, and parameters. It allows to reduce the number of parameters and computation in the network and helps to control overfitting. Just like in convolutional layers, each neuron in a pooling layer is connected to the outputs of a limited number of neurons in the previous layer, located within a small rectangular receptive field. Here, size, stride, and padding type have to be defined. However, a pooling neuron has no weights; all it does is aggregate the inputs using an aggregation function such as the max or mean. Figure 13 shows a max pooling layer, which is the most common type of pooling layer. In Figure 14, a 2×2 pooling kernel, with a stride of two and no padding, is shown. Only the max input value in each receptive field makes it to the next layer, while the other inputs are dropped. The output image has half the height and width of the input image because of the stride of two.

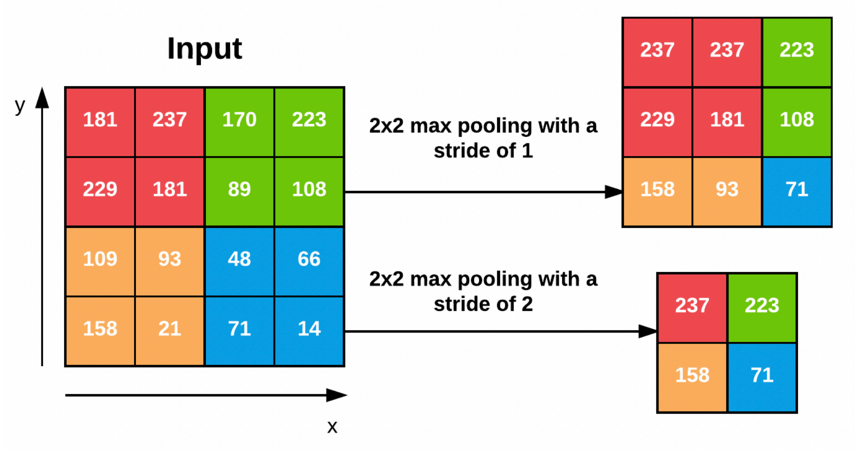


Figure 13. An example of a Pooling operation with different strides [27]

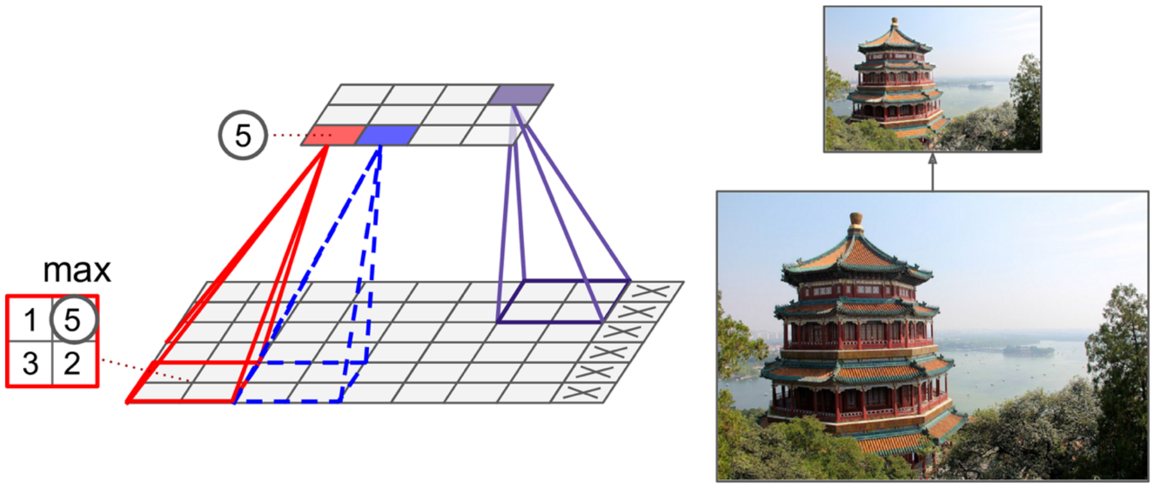


Figure 14. Max pooling layer with a 2×2 pooling kernel, stride two, and no padding

A pooling layer typically works on every input channel independently, so the output depth is the same as the input depth. Other than reducing computations, memory usage, and the number of parameters, a max pooling layer also introduces some level of invariance to small translations. By inserting a max pooling layer every few layers on a CNN, it is possible to get some translation level invariance at a larger scale. Moreover, max pooling offers a small amount of rotational invariance and small-scale invariance. Such invariance, even if limited, can be useful where the prediction should not depend on these details, such as in classification tasks. However, max pooling has some downsides as well. Firstly, it is very destructive: even with a 2×2 kernel and a stride of two, the output will be two times smaller in both directions, merely dropping 75% of the input values.

The average pooling layer works precisely like a max pooling layer, except it computes the mean rather than extracting the max value. “Average pooling layers used to be very popular, but people mostly use max pooling layers now, as they generally perform better [26].” However, max pooling preserves only the most vital features, reducing the meaningless ones, so the next layers get better-summarized features. Moreover, max pooling offers stronger translation invariance than average pooling, and it requires slightly less computation. Max pooling is typically done in the middle of the CNN architecture to reduce the spatial size, whereas average pooling is normally used as the network's final layer, such as GoogLeNet, SqueezeNet, and ResNet where it is wished to avoid using Fully-Connected layers entirely.

One last type of pooling layer that is often seen in modern architecture is the global average pooling layer. It works by computing each feature map's mean map outputting a single number per feature map and instance. Although this is extremely destructive and most of the feature map information is lost, it can be useful as the output layer.

Activation Layers

After each Convolutional layer, an Activation layer is placed in the CNN. The Activation layers are nonlinear functions such as Sigmoid, Rectified Linear Unit (ReLU), Exponential Linear Unit (ELU), and others. For the designed model, ReLU is used for the RGB-D object recognition model because it converges faster, and the slope does not saturate when *x* gets large. It does not have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh. Plots of different activation functions with their corresponding mathematical function are shown in Figure 15.

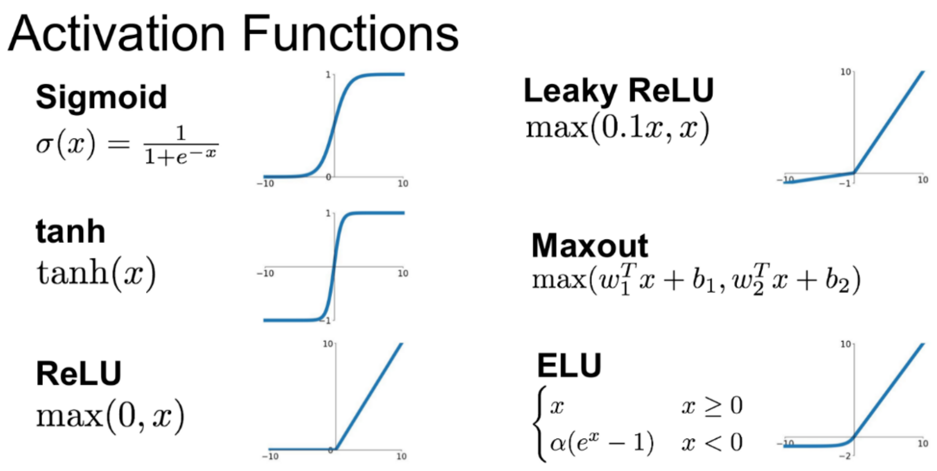


Figure 15. Plots of different activation functions

"An activation layer accepts an input volume of size *Winput*×*Hinput*×*Dinput* and then applies the given activation function, as shown in Figure 16. The activation function is applied in an element-wise manner, and the output of an activation layer is always the same as the input dimension, Winput = Woutput, Hinput = Houtput, Dinput = Doutput [27].”

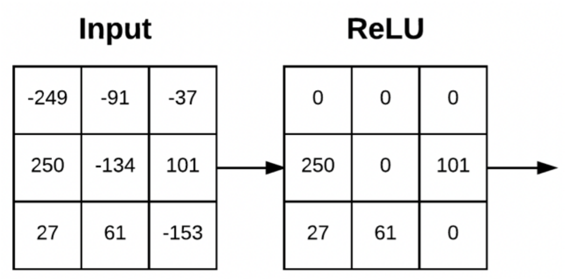


Figure 16. An example of an input volume going through a ReLU activation, max (0, x)

Fully-Connected Layers

The Fully-Connected layer is composed of neurons that are fully connected to all activations in the previous layer. Fully connected layers are always placed at the end of the network to perform classification based on extracted features from the preceding convolutional and pooling layers. This layer takes an input volume from either the convolution, ReLU, or pool layer and outputs an *N*-dimensional vector, where *N* is the number of classes that the program must realize.

Batch Normalization

Batch Normalization layers are used to normalize a given input volume activations before passing it into the next layer in the network. If x is considered as the mini-batch of activations, then the normalized $\hat{\boldsymbol{x}}$ can be computed using Eq. 2.

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}}$$

Eq. 2

Here in Eq. 2, μ_β and $\sigma^2{}_\beta$ are respectively mean and variance over each mini-batch of training images, β. The error ε is set equal to a small positive value in the range of 1e^{-7} to avoid dividing by zero. Applying implies that the activations leaving a Batch Normalization layer will have approximately zero mean and unit variance (i.e., zero-centered). During testing, the mini-batch μ_β and $\sigma^2{}_\beta$ are replaced with running averages of μ_β and $\sigma^2{}_\beta$ computed during the training process. This ensures that images through the network can be passed and obtain an accurate prediction without being biased by the μ_β and $\sigma^2{}_\beta$ from the final mini-batch passed through the network during the training phase [27].

Batch Normalization has been shown to be extremely useful for the following effects in the network.

- It reduces the number of epochs a NN takes to train itself.
- It stabilizes training by allowing for a wide variety of learning rate and regularization techniques.

- It provides a stable loss curve as it helps to minimize loss.

Dropout

Dropout is one of the most popular regularization techniques for deep neural networks. Geoffrey Hinton proposed it in 2012 [29] and further detailed in 2014 by Nitish Srivastava et al. [30], and it has proven to be highly successful. Even the state-of-the-art neural networks get a 1–2% accuracy boost by merely adding a dropout layer. When a model already has a 95% accuracy, getting a 2% accuracy boost means dropping the error rate by almost 40% by going from a 5% error to roughly a 3% error [27].

The dynamics of a dropout layer is at every training step, every neuron, including the input neurons, but always excluding the output neurons, has a probability *p* of being temporarily dropped out as shown in Figure 17. It will be entirely ignored during this training step, but it may be active during the next step. The hyperparameter *p* is called the dropout rate, and it is typically set between 10% and 50%, closer to a 20–30% in recurrent neural nets, and closer to 40–50% in CNNs. Figure 18 is shown with a 50% dropout [27]. After training, neurons do not get dropped anymore.

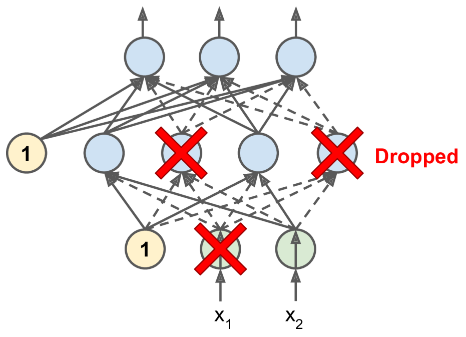


Figure 17. With dropout regularization, at each training iteration a random subset of all neurons in one or more layers except the output layer are dropped out.

Since dropout is only active during training, comparing the training loss and validation loss can be misleading. In particular, a model may be overfitting the training set and yet have similar training and validation losses. If it is observed that the model is overfitting, increasing the dropout rate can help alleviate the overfitting effect. Conversely, decrease the dropout rate if the model underfits the training set. Dropout does tend to slow down convergence significantly, but it usually results in a much better model without overfitting and underfitting when appropriately tuned.

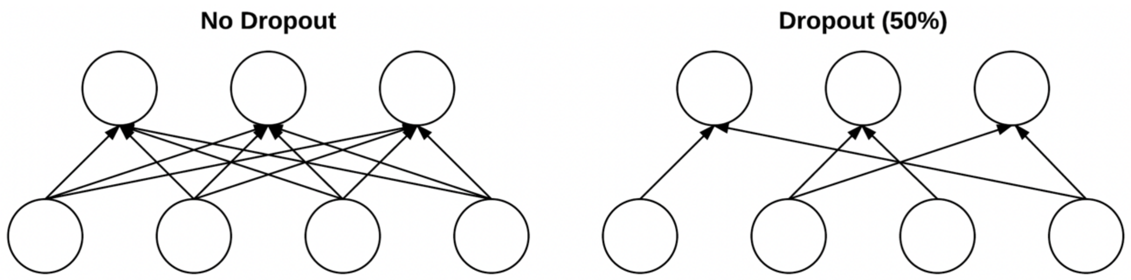


Figure 18. Effect of no Dropout (left) and Dropout with a value of 0.5 (right)

CNN Architecture

Typical CNN architectures stack a few convolutional layers, each one generally followed by an activation layer, followed by a pooling layer, then another few convolutional layers, then another pooling layer, and so on; a basic CNN architecture is shown in Figure 19. The image gets smaller and smaller as it progresses through the network, but it also typically gets deeper and deeper as the number of convolution layers increases with an increase in filter numbers. At the top of the stack, a regular feedforward neural network is added, composed of a few fully connected layers, and the final layer outputs the prediction with a Softmax. The Softmax classifier is a generalization of the binary form of the Logistic Regression layer that outputs estimated class probabilities.

When given an instance x, the Softmax Regression model first computes a score for each class, then estimates each class's probability by applying the Softmax function (or normalized exponential) to the scores.

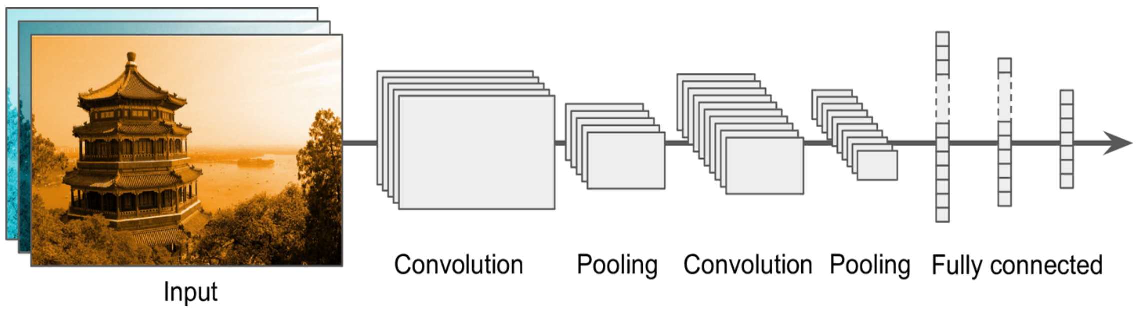


Figure 19. Typical CNN architecture [26]

In the coming chapter, Deep Convolutional Neural Network architectures are designed for object recognition for robotic grasping purposes. Further details for the designed RGB-D DCNN architecture and its training procedure are discussed.

VI. METHDOLOGY

In this chapter, the designing and optimization details of the RGB-D DCNN model architectures are discussed. This chapter is mainly divided into three parts discussing the datasets and its preprocessing, architecture design, and training of the designed network. These sections will discuss a significant role in the outcome of object recognition.

Dataset Specification

In this research work, the main goal is to identify objects in a semi-structured environment that includes poor illumination, low light, reflective surface, amorphous body, and partial object views. A large number of samples in the dataset are needed to train the designed model containing several objects in warehouse environments. Through the literature review, some datasets used for the APC competition and other purposes include blurry images, reflective images, low lights, high brightness, partial, and RGB-D images of multiple diverse objects. In this thesis, two public repositories are being used to train and test the designed model. The available datasets are mainly designed for pick-and-place robots and 3D-grasping purposes such as the CURE-OR (Challenging Unreal and Real Environment for Object Recognition) dataset [34], the MIT-Princeton dataset [4], and the RGB-D Washington dataset [5].

The datasets used for this research work are the Princeton and Washington RGB-D sample images. These datasets are developed by the MIT-Princeton University Team and Washington University, consisting of 136,575 and 41,877 RGB-D sample images. The images contain common household objects classified into 39 classes for the Princeton dataset and 51 classes for the Washington dataset. In this thesis, four classes are used:

‘*Bowl*,’ ‘*Soda can*,’ ‘*School glue,*’ ‘*Dove bar*,’ and some are shown in Figure 21. The images are captured using a RealSense F200 and Kinect style sensor, generating 640x480 resolution of RGB-D frames [4][5]. These datasets contain most of the challenging conditions which are mentioned in the research work. The machine learning model is to be trained on both public datasets containing the targeted four objects.

Pre-processing

To meet the environmental and surrounding challenges for object recognition and localization, pre-processing is required to make the model robust and applicable to all the illumination changes, different lighting conditions, and better accuracy at prediction determinations. Pre-processing increases the complexity and computation cost to the process vision solution. However, testing accuracy rates will vary and become less accurate for the challenging conditions without the pre-processing image phase. In this research, OpenCV is used for the preprocessing of the sample images, and the DCNN is used for feature extraction. These methods are compatible with Python and support machine learning libraries, including deep learning frameworks, as previously discussed in [1].

In the thesis, two preprocessing types are used; Image Resizing and change in brightness and contrast. Images of 640x480 resolution are reduced to 150x150 and 200x200 resolution for the experimentation in all the architectures with different hyperparameters and reduce the complexity and memory requirement to train the model and reduce the computation cost. The brightness and contrast are also varied randomly to perform training and testing rigorously shown in Figure 20. For contrast changes in the image samples, a random value is selected from 0 to 1, brightness value from 0 to 100.

To get the original image, the alpha parameter for contrast should be equal to 1, and the beta parameter for brightness equal to 0. For each of the models' training phase, 10,991 RGB images and 10,991 Depth images are used. For testing, 4,711 images are used for the RGB stream, and 4,711 images for the Depth stream.

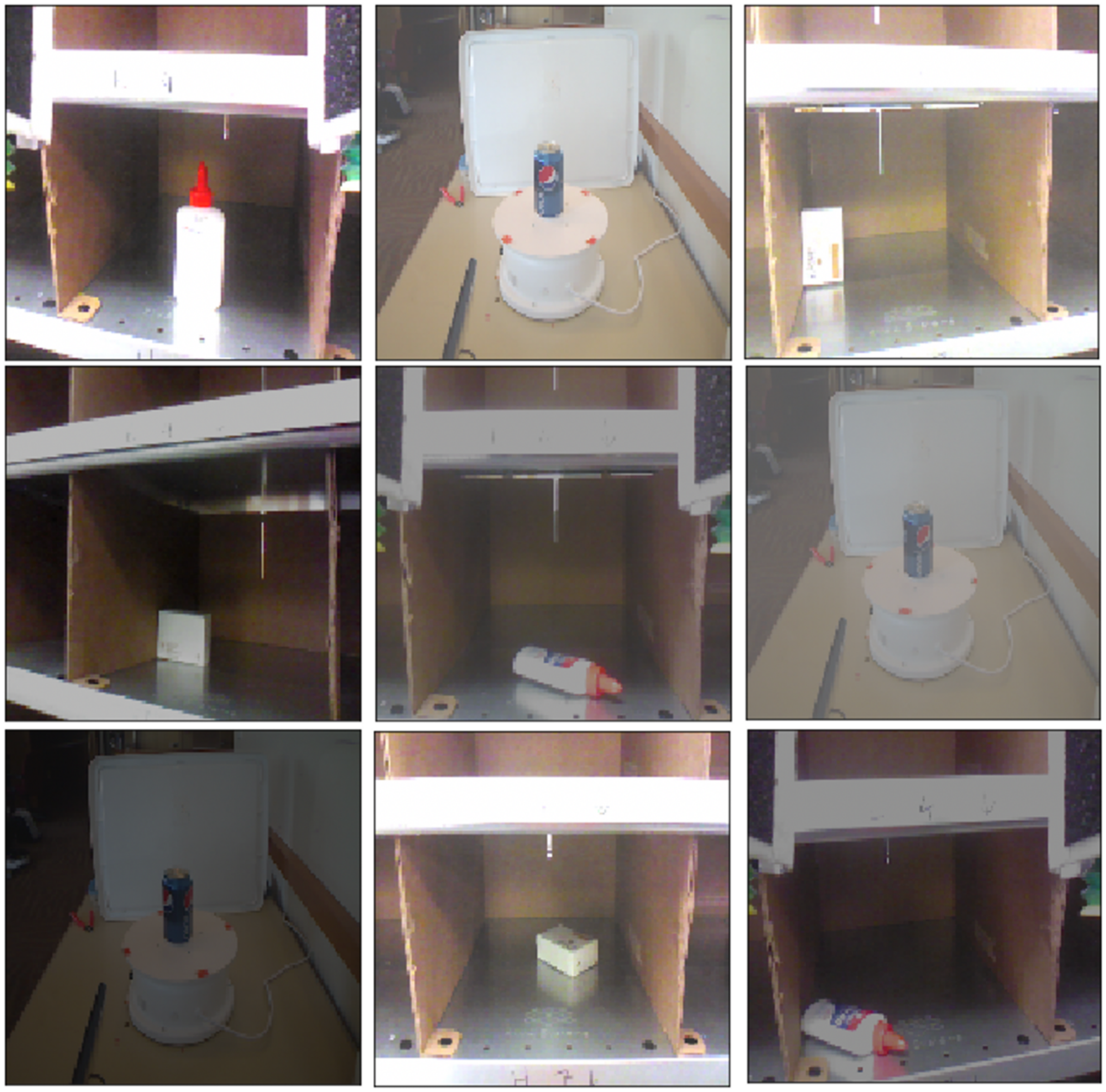


Figure 20. Some Images from dataset after preprocessing in OpenCV are shown.

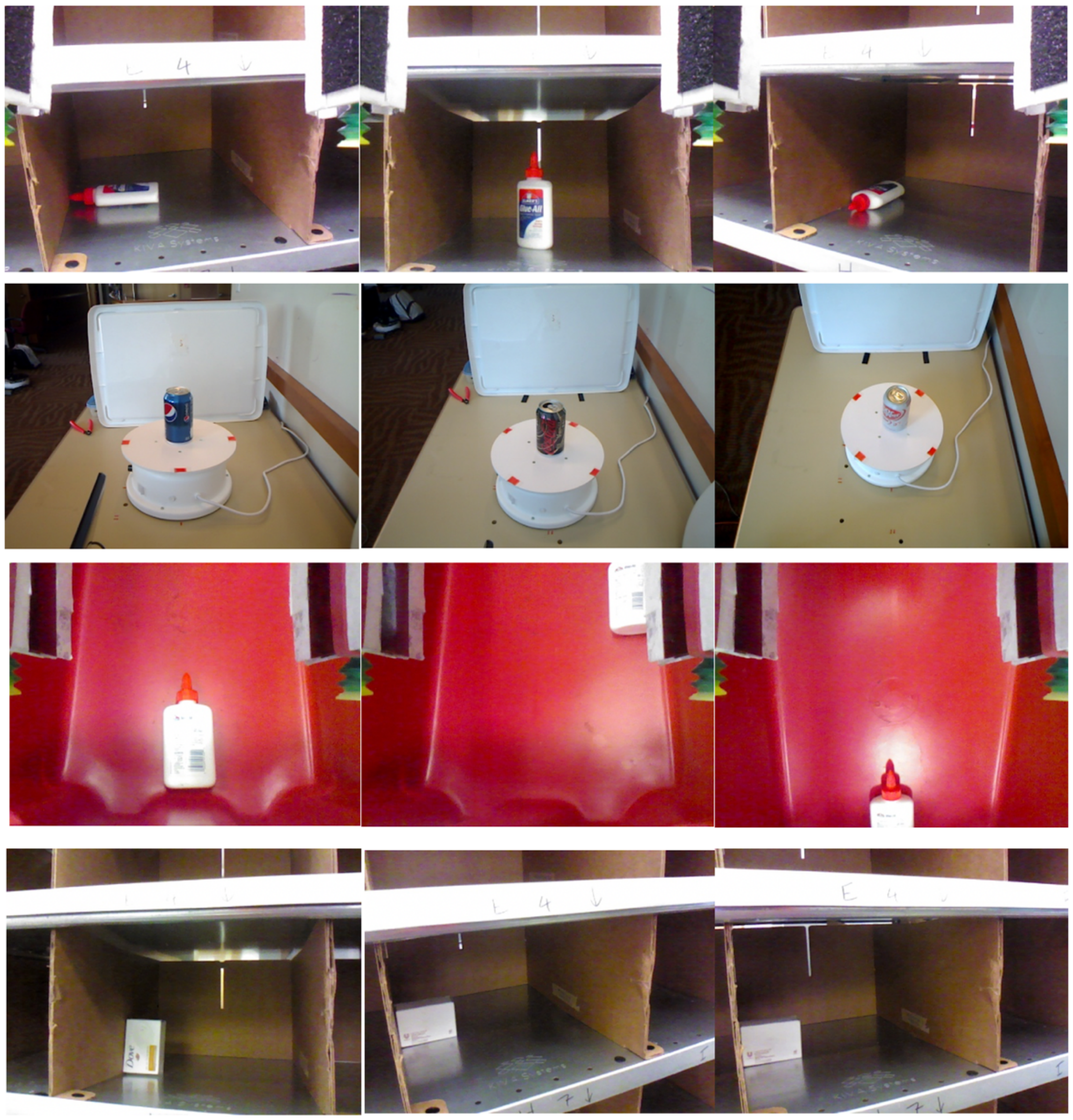


Figure 21. Image dataset with different objects.

Machine Learning Model and Base Classification Network

In the machine learning model design, DCNN is designed in the Python programming using Keras, TensorFlow open library. After the image has been pre-processed using an appropriate tool, it is served to the different layers of the DCNN model. In the CNN model, there are numbers of hidden layers which is made up of neurons. DCNN model is the best suited for image data classification and detection purposes. A simplified version of the neural network for the implemented methods is shown in Figure 22.

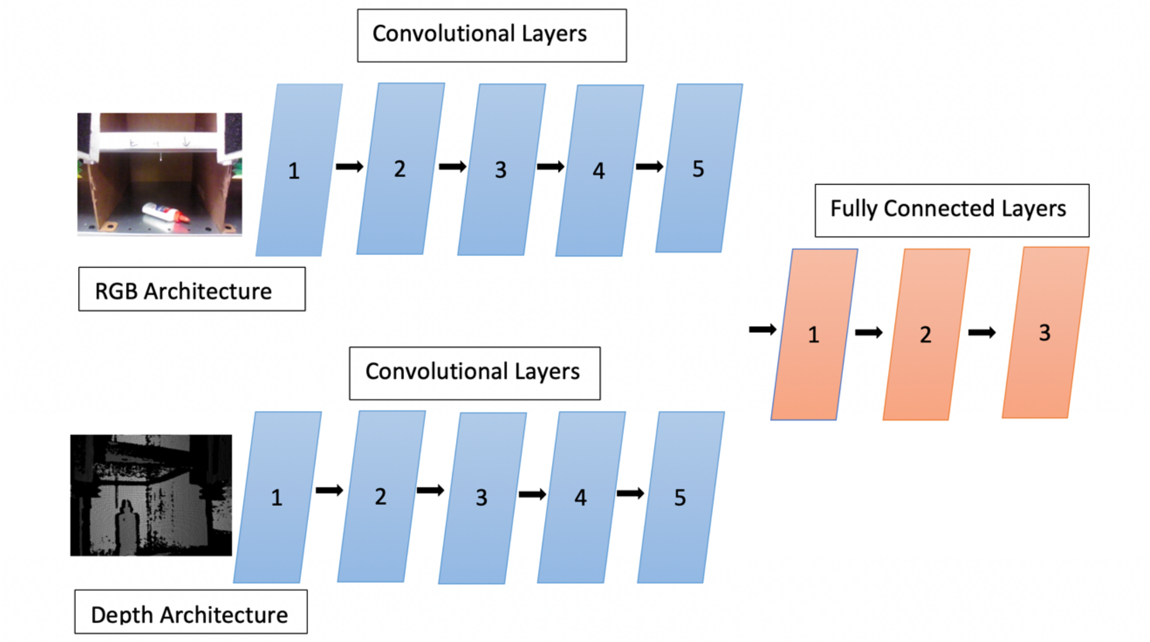


Figure.22 RGB-D DCNN Architecture [1].

To classify an object within an image, the DCNN is the deep learning technique that helps in deconstructing object features in images to classify and detect the object shapes as a singular output prediction. For the DCNN design, own architecture is made to get better results for object detection and warehousing automation for specific objects. It is desired to experiment with different DCNN models to obtain the best accuracy in detecting objects. After capturing the RGB-D images from the camera/dataset and pre-processing, each image is fed to the two parallel DCNN architecture for convolution and feature extraction shown in Figure 22 [1]. Before going to the output layer, it is required to merge the output of both DCNN. Afterward, it is connected to Fully-Connected layers, which is being used for final classification.

In the model, multiple convolutional layers are used, followed by other layers, such as the activation layer, binary normalization, and max pooling layer in a pattern for both the RGB and Depth streams. The concatenation layer is used and followed by Fully-Connected layers that classified output predicted values using a Softmax layer to combine the RGB and Depth network streams

First Designed RGB-D DCNN model

In this section, details of architecture I are discussed, how it is designed using different layers and their parameters, and a complete table of the parameters are shown in Table 1.

- The first convolutional layer uses eight fairly large filters (7×7). It also sets input shape= [150, 150, 3] for RGB, and input shape of [150, 150, 1] for Depth image because the images are 150 × 150 pixels after pre-processing, with a three- and single-color channel (i.e., RGB = 3, Grayscale = 1). After this layer, it is followed by the same convolutional layer to learn more features from the previous layer; it can be seen in Table 1 Parameter section that parameters increased.
- A max pooling layer uses a pool size of 2 and strides of 1. It divides each spatial dimension by a factor of 2. The Pool layer's primary function is to progressively reduce the spatial size, width, and height, of the input volume. Doing this reduces the number of parameters and computation in the network pooling and helps control overfitting.
- Dropout layer with a dropout value of 0.5 to reduce overfitting by explicitly altering the network architecture at training time
- The structure is repeated with two convolutional layers followed by a max pooling layer with some changes in the convolutional layer's filter size from (7x7) to (3x3) and changes in the dropout layer from 0.5 to 0.2. Again, one convolutional layer is followed by a max pooling layer. The number of filters grows as CNN climbs up toward the output layer (it is initially 8, then 16, then 32): it makes sense to grow since the number of low-level features is often

relatively low (e.g., small circles, horizontal lines), but there are many different ways to combine them into higher-level features. It is a common practice to double the number of filters after each pooling layer: since a pooling layer divides each spatial dimension by a factor of 2, double the number of feature maps in the next layer can be afforded without fear of exploding the number of parameters, memory usage, or computational load.

- The Fully-Connected network is composed of flattening layers and one dense layer. The inputs must be flattened since a dense network expects a 1D-array of features for each instance. A Dropout layer is added, with a dropout rate of 20%, to reduce overfitting and regularization of weight adjustment during each model's training process.
- Similarly, for the Depth architecture. Next is combining both the architectures using a concatenation layer followed by two more dense layers. Finally, the probability of each class and training loss is calculated by the Softmax layer.
- The number of parameters is increased due to the increase in the number of filters in the convolution layer. The learnable parameters can be calculated by multiplying the shape of width, height, number of filters from the previous layer plus one, and the whole equation multiplied with the current layer's filters. The number of parameters in a first CONV layer in Table 1 would be: ((shape of the width of the filter * shape of the height of the filter * the number of filters in the previous layer+1) * the number of filters in the current layer). For first layer filter (shape = (7x7), stride=1) layer is: ((shape of width of filter * shape of height filter

* the number of filters in the previous layer+1) * the number of filters) = (((7*7*3) +1) *8) = 1184.

- Thus, the Fully-Connected layer's number of parameters can be calculated: ((current layer neurons c * previous layer neurons p) +1 * c). For the Softmax layer, (1024*4+ 1*4) = 4100. The total trainable parameters in this architecture are 546,369,404.

Table.1 Architecture I for RGB-D DCNN model

Layers	RGB Architecture		Depth Architecture	
	Output Shape	Parameter	Output Shape	Parameter
Input	150 x 150 x 3	0	150 x 150 x 1	0
Conv	144 x 144 x8	1184	144 x 144 x 8	400
Conv	138 x 138 x 8	3144	138 x 138 x 8	3144
MaxPool	137 x 137 x 8	0	137 x 137 x 8	0
Dropout	137 x 137 x 8	0	137 x 137 x 8	0
Conv	135 x 135 x 8	584	135 x 135 x 16	1168
Conv	133 x 133 x 16	1168	133 x 133 x 16	2320
MaxPool	132 x 132 x 16	0	132 x 132 x 16	0
Dropout	132 x 132 x 16	0	132 x 132 x 16	0
conv	130 x 130 x 32	4640	130 x 130 x 32	4640
MaxPool	129 x 129 x 32	0	129 x 129 x 32	0
Dropout	129 x 129 x 32	0	129 x 129 x 32	0
Flatten	532512	0	532512	0
Dense	512	272646656	512	272646656
Dropout	512	0	512	0
Concatenation	None, 1024			0
Dense	None , 1024			1049600
Dropout	None , 1024			0
Softmax	None, 4			4100
Total Parameter:	**546,369,404**			
Trainable Parameter:	**546,369,404**			

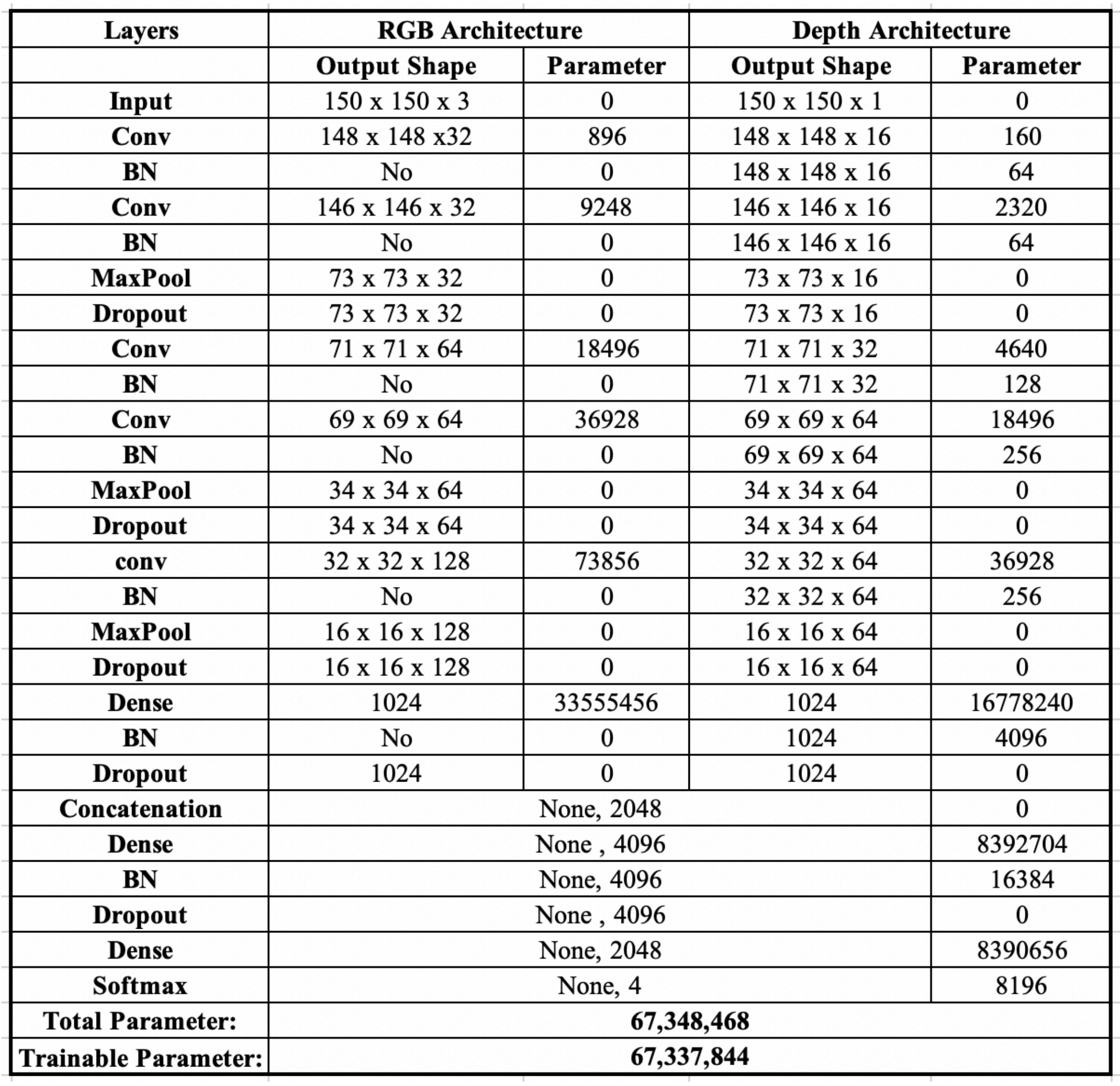
Second Designed RGB-D DCNN model

In architecture II, there are some changes compared to architecture I to improve the results and accuracy.

- The first convolutional layer uses 32 small filters (3×3). Here, compared to previous architecture, some parameters are changed, such as filter size and dropout. The number of filters used in this architecture for the RGB stream is 32, 64, 128, and for the Depth stream, 16, 32, and 64. The features extracted with the filter size of (3x3) are highly local, capturing smaller, complex features in the image.
- The batch normalization layer is also used in the depth architecture, which is not used in the previous architecture. It makes the learning rate and regularization less volatile and more straightforward to tune. It reduces final loss and a more stable loss curve when using batch normalization in the networks.
- The dropout values are kept 0.3 in both the stream, but just before Concatenation and after Concatenation, it increased to 0.5 as it was dealing with a vast amount of parameters.
- Number of parameters in a first Conv layer in Table 2 would be; filter shape = (3x3), stride=1) layer is: ((the shape of the width of filter * the shape of the height filter * the number of filters in the previous layer+1) * the number of filters) = (((3*3*3) +1) *32) = 896.
- In this architecture, it can be seen in Table 2 that the number of parameters is more in the last Softmax layer, which can be calculated as mentioned in

architecture I. Here, total trainable parameters are 64,337,844, which are less compared to architecture I making it train faster.

Table.2 Architecture II for RGB-D DCNN model



Layers	RGB Architecture		Depth Architecture	
	Output Shape	**Parameter**	**Output Shape**	**Parameter**
Input	150 x 150 x 3	0	150 x 150 x 1	0
Conv	148 x 148 x32	896	148 x 148 x 16	160
BN	No	0	148 x 148 x 16	64
Conv	146 x 146 x 32	9248	146 x 146 x 16	2320
BN	No	0	146 x 146 x 16	64
MaxPool	73 x 73 x 32	0	73 x 73 x 16	0
Dropout	73 x 73 x 32	0	73 x 73 x 16	0
Conv	71 x 71 x 64	18496	71 x 71 x 32	4640
BN	No	0	71 x 71 x 32	128
Conv	69 x 69 x 64	36928	69 x 69 x 64	18496
BN	No	0	69 x 69 x 64	256
MaxPool	34 x 34 x 64	0	34 x 34 x 64	0
Dropout	34 x 34 x 64	0	34 x 34 x 64	0
conv	32 x 32 x 128	73856	32 x 32 x 64	36928
BN	No	0	32 x 32 x 64	256
MaxPool	16 x 16 x 128	0	16 x 16 x 64	0
Dropout	16 x 16 x 128	0	16 x 16 x 64	0
Dense	1024	33555456	1024	16778240
BN	No	0	1024	4096
Dropout	1024	0	1024	0
Concatenation	None, 2048			0
Dense	None , 4096			8392704
BN	None, 4096			16384
Dropout	None , 4096			0
Dense	None, 2048			8390656
Softmax	None, 4			8196
Total Parameter:	**67,348,468**			
Trainable Parameter:	**67,337,844**			

Third Designed RGB-D DCNN model

In this section of designing architecture, more changes in architecture III are discussed:

- Architecture III has a different pattern for convolution layers compared to the previous two architectures. In this design, the first group of layers contains three convolution layers with the filter size of (3x3) to the amount of information or features extracted will be huge, and it learns complex feature, which can be further useful in later layers. Convolution layers are followed by batch normalization and dropout. The dropout values are kept similar to the architecture II
- In the second group of layers, two convolution layers are used, followed by batch normalization and max pool.
- Next layer contains one convolution layer, then similarly batch normalization and max pool. For combining two networks concatenation layer is used to combine the information from both the streams, similarly mentioned in architecture I.
- Number of parameters in a first CONV layer in Table 3 would be: ((shape of the width of the filter * the shape of the height of the filter * the number of filters in the previous layer+1) * the number of filters in the current layer). For first layer filter shape = (3x3), stride=1) layer is: ((shape of width of filter*shape of height filter*number of filters in the previous layer+1) *number of filters) = (((3*3*3) +1) *16) = 448.

- The total number of trainable parameters obtained in this architecture are 475,721,892. As can be seen, there are many parameters to train, which increases computation complexity and memory requirement.

Table.3 Architecture III for RGB-D DCNN model

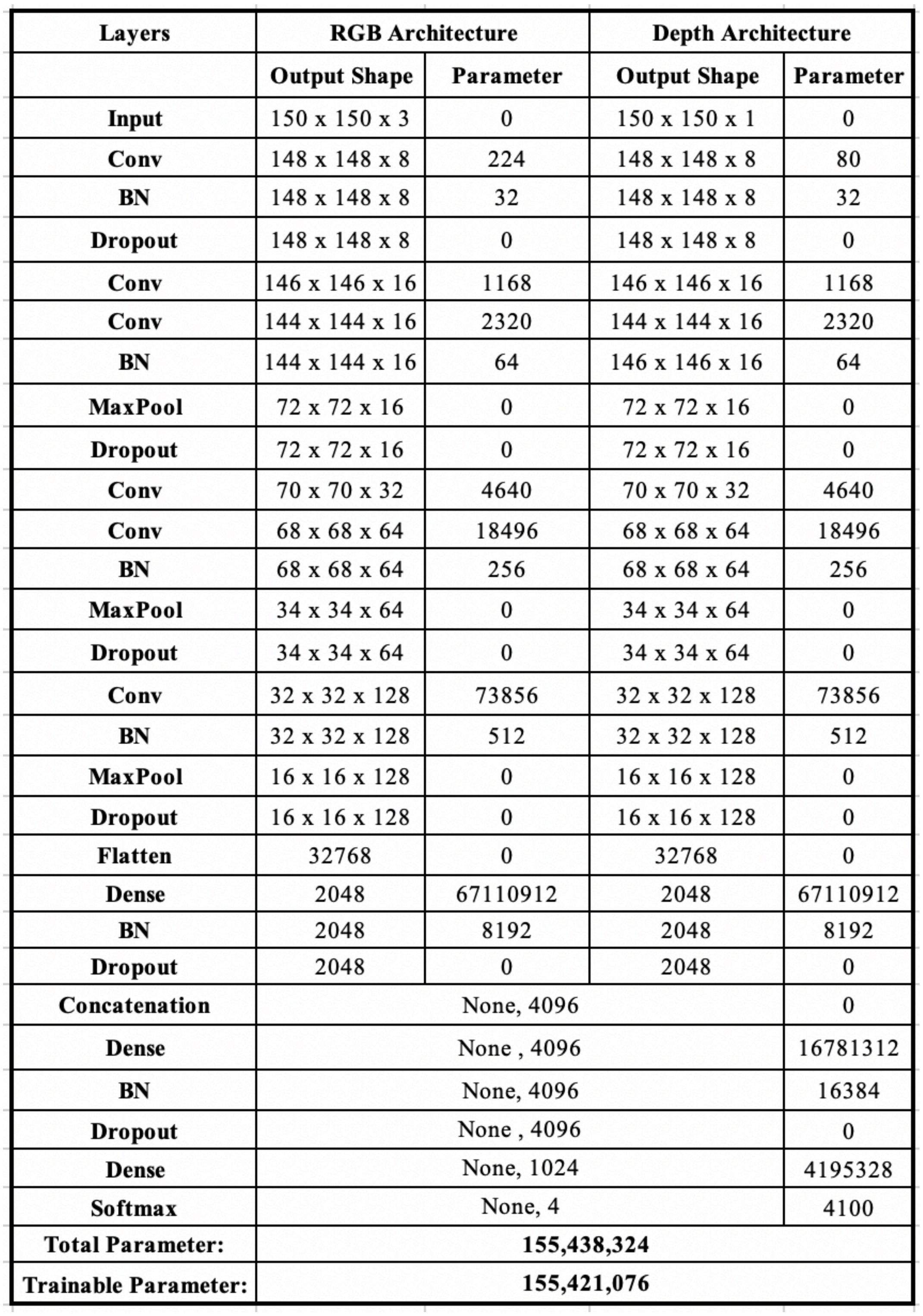
Layers	RGB Architecture		Depth Architecture	
	Output Shape	Parameter	Output Shape	Parameter
Input	150 x 150 x 3	0	150 x 150 x 1	0
Conv	148 x 148 x 16	448	148 x 148 x 16	160
Conv	146 x 146 x 16	2320	146 x 146 x 16	2320
Conv	144 x 144 x 16	2320	144 x 144 x 16	2320
BN	144 x 144 x 16	64	144 x 144 x 16	64
Dropout	144 x 144 x 16	0	144 x 144 x 16	0
Conv	142 x 142 x 64	9280	142 x 142 x 32	4640
Conv	140 x 140 x 64	36928	140 x 140 x 32	9248
BN	140 x 140 x 64	256	140 x 140 x 32	128
MaxPool	70 x 70 x 64	0	70 x 70 x 32	0
Dropout	70 x 70 x 64	0	70 x 70 x 32	0
conv	68 x 68 x 128	73856	68 x 68 x 64	18496
BN	68 x 68 x 128	512	68 x 68 x 64	256
MaxPool	34 x 34 x 128	0	34 x 34 x 64	0
Dropout	34 x 34 x 128	0	34 x 34 x 64	0
Flatten	147968	0	73984	0
Dense	2048	303040512	2048	151521280
BN	2048	8192	2048	8192
Dropout	2048	0	2048	0
Concatenation	None, 4096			0
Dense	None , 4096			16781312
BN	None, 4096			16384
Dropout	None , 4096			0
Dense	None, 1024			4195328
Softmax	None, 4			4100
Total Parameter:	**475,738,916**			
Trainable Parameter:	**475,721,892**			

Fourth Designed RGB-D DCNN model

Architecture IV design is different from the other architectures mentioned above by using six convolution layers in each RGB and Depth network to make the network deeper and make it stronger to learn all the low-level features with the increase in the number of filters. It has more convolution layers, but the total number of parameters is less than architecture I and architecture III because the max pool layer used in this with the stride size is (2, 2), which reduces the output size by 2. Filters used in this architecture with the first Convolution layer are eight, with the second and third convolution layer 16, with fourth and fifth it is 32 and 64, with the last sixth Convolution layer it is 128. Similarly, used for the Depth stream.

As it is discussed in the previous architectures, the number of parameters in a first CONV layer in Table 4 would be: ((width of the filter * height of the filter * the number of filters in the previous layer+1) * the number of filters in current layer). For first layer, the filter shape = (3x3), stride=1) layer is: ((width of the filter * height of the filter * the number of filters in the previous layer+1) * the number of filters) = (((3*3*3) +1) *8) = 224. Similarly, parameters can also be calculated for depth stream and other convolution layers in the RGB and the Depth stream. Here, the total number of parameters is 155,421,076. Total parameters are less than architecture I and architecture II, making it faster in training than these two architectures.

Table.4 Architecture IV for RGB-D DCNN model



Layers	RGB Architecture		Depth Architecture	
	Output Shape	Parameter	Output Shape	Parameter
Input	150 x 150 x 3	0	150 x 150 x 1	0
Conv	148 x 148 x 8	224	148 x 148 x 8	80
BN	148 x 148 x 8	32	148 x 148 x 8	32
Dropout	148 x 148 x 8	0	148 x 148 x 8	0
Conv	146 x 146 x 16	1168	146 x 146 x 16	1168
Conv	144 x 144 x 16	2320	144 x 144 x 16	2320
BN	144 x 144 x 16	64	146 x 146 x 16	64
MaxPool	72 x 72 x 16	0	72 x 72 x 16	0
Dropout	72 x 72 x 16	0	72 x 72 x 16	0
Conv	70 x 70 x 32	4640	70 x 70 x 32	4640
Conv	68 x 68 x 64	18496	68 x 68 x 64	18496
BN	68 x 68 x 64	256	68 x 68 x 64	256
MaxPool	34 x 34 x 64	0	34 x 34 x 64	0
Dropout	34 x 34 x 64	0	34 x 34 x 64	0
Conv	32 x 32 x 128	73856	32 x 32 x 128	73856
BN	32 x 32 x 128	512	32 x 32 x 128	512
MaxPool	16 x 16 x 128	0	16 x 16 x 128	0
Dropout	16 x 16 x 128	0	16 x 16 x 128	0
Flatten	32768	0	32768	0
Dense	2048	67110912	2048	67110912
BN	2048	8192	2048	8192
Dropout	2048	0	2048	0
Concatenation	None, 4096			0
Dense	None , 4096			16781312
BN	None, 4096			16384
Dropout	None , 4096			0
Dense	None, 1024			4195328
Softmax	None, 4			4100
Total Parameter:	**155,438,324**			
Trainable Parameter:	**155,421,076**			

Training the Designed DCNN model

A backpropagation algorithm is used to train the designed DCNN. This algorithm is an efficient method to calculate the gradient of the loss function. It gives the details of changes of the weights and bias of the parameters in the network. Gradient descent is the most common way of optimizing CNN loss functions. The procedure of repeatedly evaluating the gradient of a loss function and performing a parameter update is known as the gradient descent algorithm. In this research, the DCNN model is trained from scratch (with random initialization) instead of taking a pre-trained network; it uses the gradient descent technique.

There are three different types of gradient descent techniques.

- **Batch gradient descent**: It computes the gradient of the cost function for the entire training dataset. It calculates the gradients in one step for the complete dataset to perform an update. This method can be prolonged and expensive in terms of memory.

- **Stochastic gradient descent (SGD)**: It computes the gradient of the cost function for one training sample at each step. Therefore, SGD performs a parameter update for each training sample with a high variance. It causes the loss function to fluctuate heavily, which allows it to reach new and potentially better local minima. The SGD is usually much faster since it updates weights frequently at each step.

- **Mini-batch gradient descent**: It computes the gradient of the cost function in a small batch. Therefore, an update is performed for every mini-batch of *n* training samples. This approach reduces the variance of the parameter updates, which can lead to more stable convergence. Mini-batch gradient descent is typically the algorithm of choice when training CNN.

Several more optimization methods available for deep learning training, such as Adagrad, RMSprop, Adadelta, and Adam techniques. For the model to learn faster and reduce the training time, the mini-batch gradient descent with momentum was used. Momentum methods are normally used to accelerate the learning to fast convergence and allow models to obtain lower loss, higher accuracy, and a reduced number of epochs. Mini-batch gradient descent continues to be one of the most utilized in model design as the primary optimization method.

Regularization

It is defined as *"any method that increases testing accuracy perhaps at the expense of training accuracy"* [27]. Regularization is a mechanism to prevent the model from overfitting and provides the ability to generalize the model better. There are various types of regularization techniques, such as L1 regularization, L2 regularization, commonly called "weight decay." During experimentation and optimization of the model, no regularization is added to the model; however, the dropout layer is used as a regularization to improve the model's classification accuracy. Additionally, some data augmentation techniques are also used to provide the model more flexibility to unobserved data, reduce overfitting, and enhance the ability to generalize.

Hyperparameter Tuning

In this section, hyperparameters are discussed and are responsible for the tuned performance of a model. The hyperparameters in the DCNN architectures that were used in this research are discussed in detail below.

Learning Rate

The learning rate is one of the essential hyperparameter of the deep neural network. It is the parameter that decides the rate or speed at which the model learns. Finding a perfect learning rate is difficult, but the most significant task, it is not possible to fix a single learning rate for every model and dataset. If the learning rate is too low, the training will be prolonged, but at some point, it will converge to the optimum. If the learning rate is slightly too high, it may diverge or end up at around the optimum and converge in its learning process. There are different strategies to get the best learning rate; one of the best-known techniques is the learning scheduler. In this research, experimentation is done at different rates to get the most optimized learning rate value. The learning rate scheduler is beneficial when training deep neural networks, where the learning rate can be changed or scheduled at training time. In this work, the standard learning rate schedulers available in the Keras library are used. This scheduler decreases the learning rate gradually after every batch update, and it is controlled by applying the following formula for adjusting the learning rate after every batch update:

$$lr = initial\_lr \times (1/(1 + decay \times iteration)) \qquad \text{Eq.1}$$

Herein Eq.1, *lr* is the learning rate, and the initial learning rate is the parameters tuned in the process during training. Iterations can be calculated as the total number of training images to the batch size. The decay can be started with the initial learning rate to the number of epochs. In Figure 23, the learning curves are shown with the different learning rates as the number of epochs increases. These curves can be referred to and compared with the results to get a better learning rate.

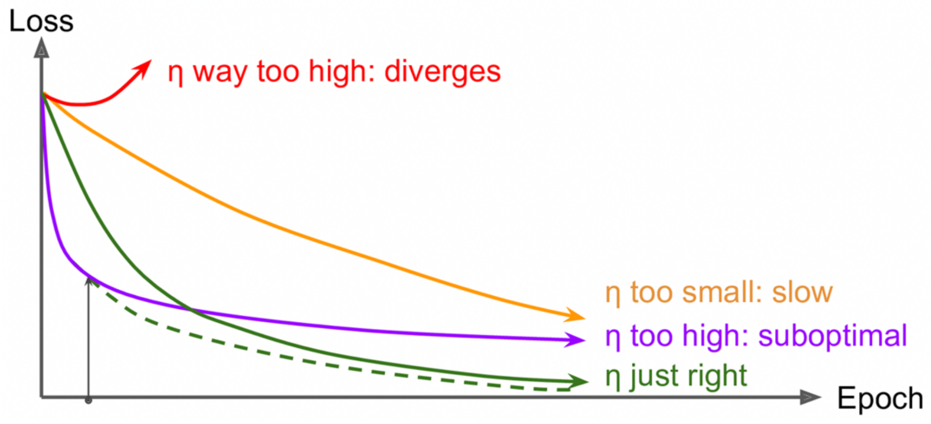


Figure 23. Learning curves for various learning rates(η)

Number of epochs

The number of epochs is the number of times the whole training dataset passes through the neural network. The accuracy graph plots are mostly loss (or accuracy) vs. the number of epochs to analyze the model performance. The number of epochs is changed after analyzing accuracy curves to overcome the underfitting and overfitting problem shown in Figure 24. Throughout the experiments, the number of epochs was set to 50, 60, 70, 80, 100, and 200.

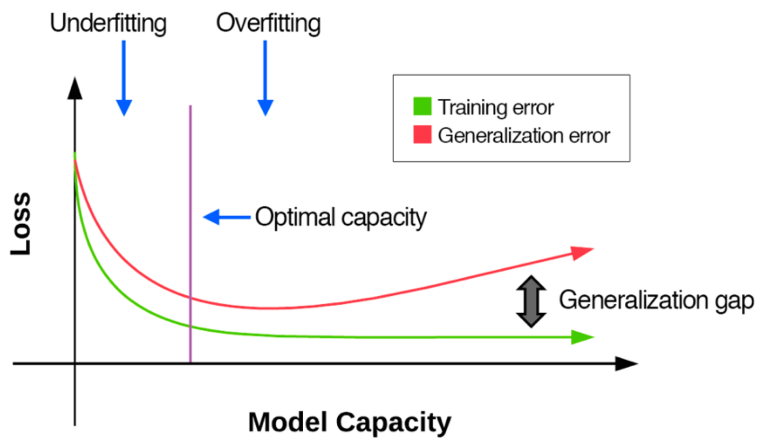


Figure 24. Relationship between model capacity and loss

Batch size

The batch size is used in gradient descent to reduce the variance of the parameter updates, leading to a stable convergence. Generally, mini-batch sizes range between 16 and 256 but can vary for different applications. The model is tested with batch sizes of 32, 64, and 128. Performance and accuracy can be changed with the batch size to get the best results.

Momentum

The momentum hyperparameter controls gradient descent's acceleration in the relevant direction and in dampening the oscillation shown in Figure 25. It is set to a value greater than zero and less than one, where shared values such

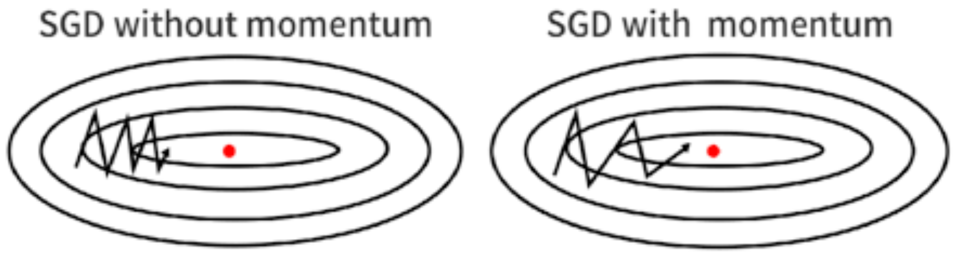


Figure 25. SGD without (on left) and with momentum (on right)

as 0.9 [27]. In the designed model, momentum is set to 0.9 to reduce the oscillations.

This chapter details the dataset, dataset preprocessing with OpenCV, and the designing of all four architectures with the components needed for training a DCNN. The loss function, and optimization methods, and an explanation of each layer of the model architecture were also covered. In the next chapter, the designed architecture results are discussed in the details with the accuracy and losses curves.

VII. RESULTS AND DISCUSSION

In this section, the training and test results with different architectures and datasets are discussed. Different experiments were performed by changing the hyperparameters of different architectures, shown via plots and tables for the available datasets to understand the model's accuracy effects. Also, the computational resources used in this thesis for training the CNN are discussed here.

Computational Resources

In this thesis, all the DCNN architectures were designed using the Python programming language version 3.7.3 using Anaconda. It is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, and other applications) [36]. For Python programming, open-source libraries are available for deep learning, such as TensorFlow, Keras (backend of TensorFlow), which are popular libraries that support the building of DCNNs. The training and evaluation of DCNN models can be prolonged due to the computations required for each iteration in training. For training the DCNN model, the LEAP (Learning, Exploration, Analysis, and Processing) next-generation High-Performance Computing (HPC) Cluster of Texas State University was used. The cluster's head-node and 120 compute-nodes are configured with Dell PowerEdge C6320 server nodes, each with 28 CPU cores via two (14-core) 2.4 GHz E5-2680v4 Intel Xeon (Broadwell) processors. With 128GB of memory and 400GB of SSD storage per node, the compute-nodes provide an aggregate of 15TB of memory and 48TB of local scratch storage. Additionally, the LEAP cluster features two large memory (1.5TB of RAM) nodes with 64 CPU cores via four (16-core) 2.5 GHz E7-8867v3 Intel Xeon (Haswell) processors. Figure 26 shows the features and components of the LEAP cluster.

Apart from LEAP Cluster, the HiPE Server was also used by the High-Performance Computing research group at the Ingram School of Engineering. The HiPE3 server is a Dell PowerEdge R740 rack server configured with dual 2.3 GHz Intel Xeon-Gold with 18 Cores/36 thread capacity [37]. The server has 16GB RDIMM x12 Data-width (192GB) of memory and dual 1.2TB Solid State Drive SATA storage. Using the HiPE3 server for training the CNN has sped up the process because of the memory allocation and low congestion. The HiPE3 server is also equipped with two NVIDIA Tesla V100 accelerators.

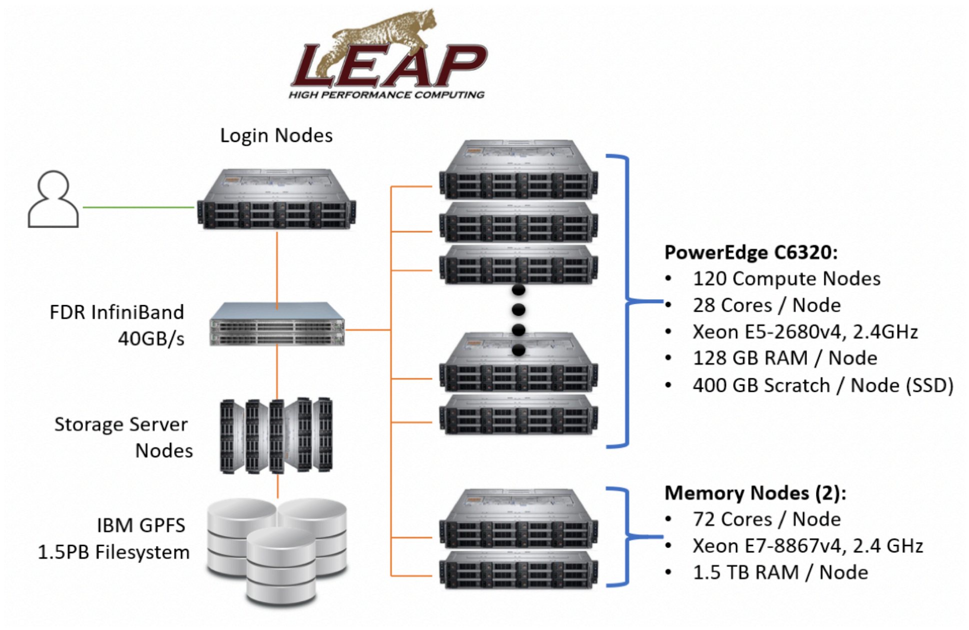


Figure 26. Features and components of LEAP cluster

Training and Testing Results

In this research, the main focus is to design and optimize the RGB-D Architecture for the 3D-Detection base network to get better accuracy for partial images and low-illumination images. Experimentation was performed on four different architecture to get the most optimized and efficient architecture with reliable accuracy.

Initially, the hypothesis was to design a single architecture and better classify object detection accuracy. However, as research progressed, four architecture designs with variation in layers and other parameters were designed and tested on the dataset. The experiments were performed with the pre-processing for different lighting conditions using OpenCV for changing the brightness and contrast to get better testing accuracy for recognition. The model is tested and optimized further for better results for grasping purposes.

Different combinations of hyperparameters were tested to get a final reliable model with good accuracy for testing the dataset with four different objects ‘*Bowl*,’ ‘*Soda can*,’ ‘*School glue*,’ ‘*Dove bar*.’ Different parameters, such as learning rate, epochs, batch size, image resolution, varied in multiple experiments. The obtained results were compared and analyzed to get a better model. As mentioned previously, the LEAP cluster and HiPE server were used for all the experiments.

Performance of Architecture I

Experimentation on architecture I started without pre-processing with 200 epochs for 100x100 image resolution. After observing the learning curves, it was realized that 200 epochs are more suitable to train and improve the performance of the model. In the beginning, the dropout value, number of filters, and size of filters were also tested to observe the change of the learning pattern with each change made. In this first experiment, a 0.0001 learning rate was chosen as an initial point with a batch size of 64. The reason behind selecting this low learning rate value is to observe the learning curve, time to learn, and to reach an optimal loss. Further, number of epochs were reduced to 140, 100, 80, with different batch sizes of 32, 64, and 128, and learning rates of 0.01, 0.001, and 0.0001 were also tested.

In the different hyperparameters settings for architecture I, it was observed in Figure 27 that the model was learning very fast, even after keeping the learning rate at 0.0001 without pre-processing the 100x100 image for the brightness and contrast. In this case, it will be hard to train for new images, and it would not be able to learn new features. For improvement in the training learning curves, resolution of images was changed to the 150x150 and epochs were reduced to 140, but no improvement observed, and the architecture was learning too fast shown in Figure 28. Figure 29 show the pre-processing effect for the changes in brightness and contrast to add variations in the images during the training of the model, with a learning rate reduced to 0.001, and 80 epochs to observe the model accuracy behavior to the variations. The batch size changed to 128 in Figure 30 to see if the problem persists with the larger batch size to keep improving the accuracy performance. Figure 31 shows the model's accuracy curve with a batch size of 32, image resolution of 200x200 and learning rate of 0.0001. Apart from all these experiments, the model also tested with the change in the number of filters and dropout values. For this architecture, the time taken to train the model was around ten days on the LEAP cluster. It was concluded with these results that this architecture model is not suitable for object recognition purposes.

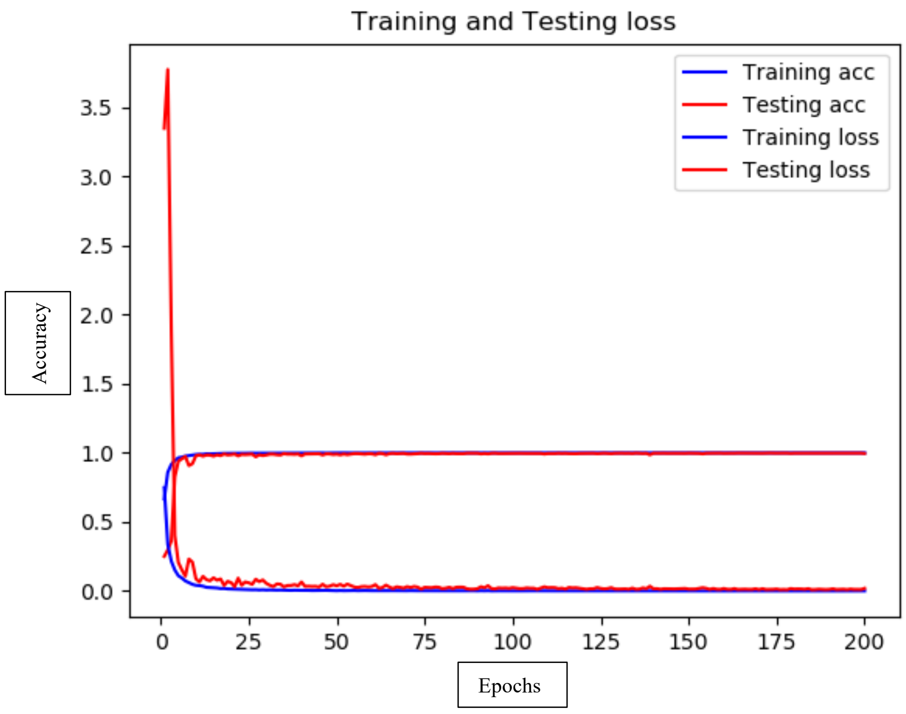


Figure 27. Initial result obtained from architecture I with *lr* 0.0001, for a batch size of 64, and without pre-processing with a 100x100 image resolution

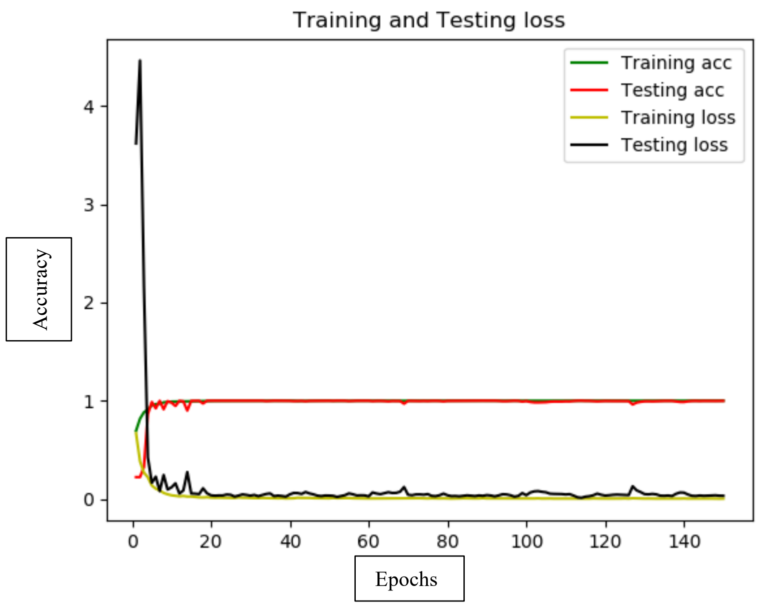


Figure 28. Initial result obtained from architecture I with *lr* 0.0001, for a batch size of 64, and without pre-processing with 140 epochs with a 150x150 image resolution

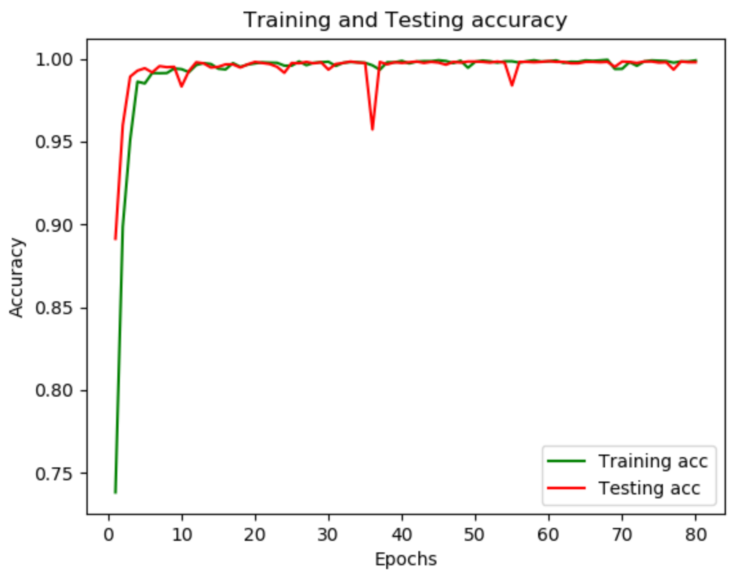


Figure 29. Accuracy curve with a *lr* of 0.001, for a batch size of 64 with 80 epochs, and with pre-processing with a 150x150 image resolution

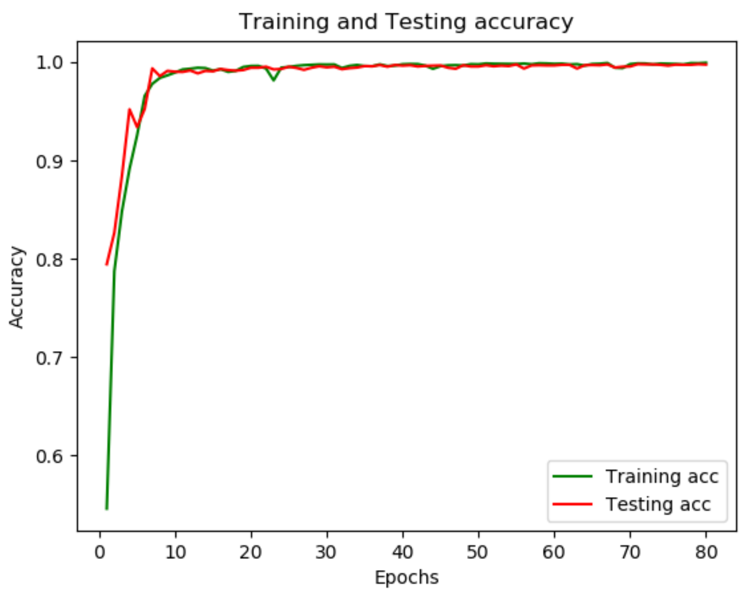


Figure 30. Accuracy curve with a *lr* of 0.001, for a batch size of 128 with 80 epochs, and with pre-processing with a 150x150 image resolution

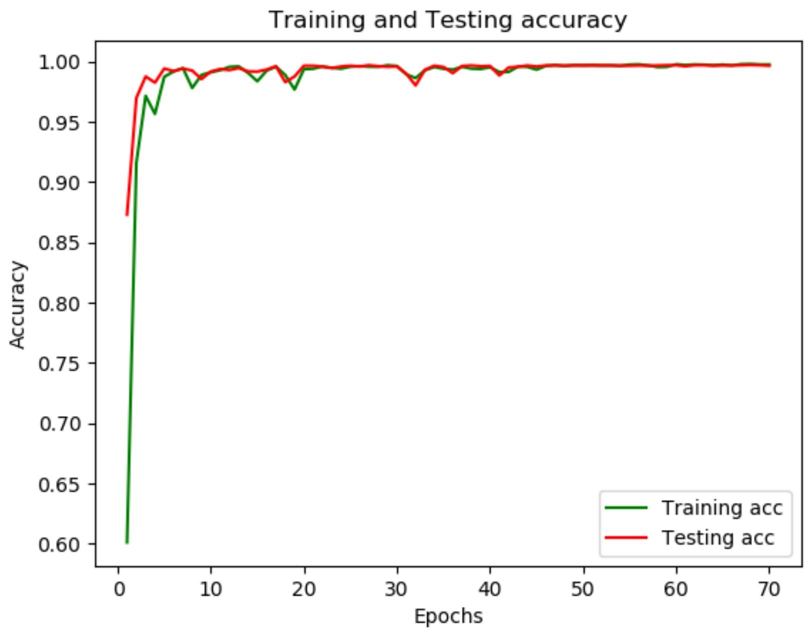
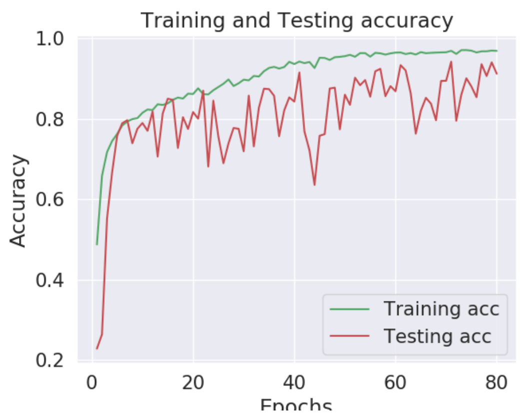


Figure 31. Accuracy curve with *lr* 0.0001 for a batch size of 32 with 70 epochs, and with preprocessing with a 200x200 image resolution

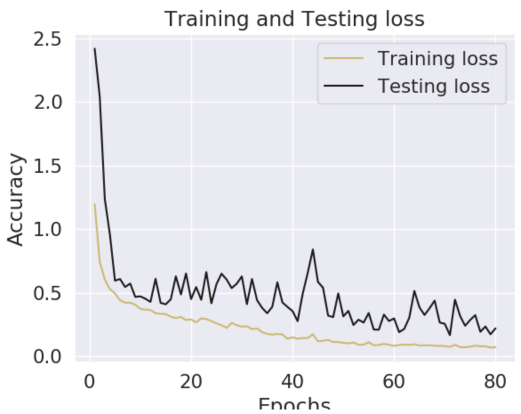
Performance of Architecture II

Experimentation procedure performed on architecture II was similar to the architecture I. Architecture II performed well compared to other architectures in the speed and accuracy. It took less time to train the model compared to other models because of its design and layers used in it. On the LEAP server time taken by architecture II was approximate eight days to complete training for on an approximate 80 epochs.

In Figure 32, experimental results with the *lr* of 0.0001 with a batch size of 64 with the 80 epochs of 150x150 image size are shown. The learning curves obtained from this architecture are much better and improved. Here, the testing accuracy curve was obeyed the training accuracy curve. Accuracy is improving with the increase in epoch without overfitting can be seen in the loss curve in Figure 32 b). The accuracy obtained with this architecture is 91.23%.



a)



b)

Figure 32. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 64 with 80 epochs, and with pre-processing with a 150x150 image resolution

In Figure 33, experiments were performed with *lr* of 0.0001 for a batch size of 128 with the image resolution of 150x150. In the accuracy curve, it is observed that it was not easy to predict test image, and sometimes testing accuracy is more than the training image, which tells that testing images were more comfortable to predict on few occasions. The test accuracy obtained by this is 96.69%.

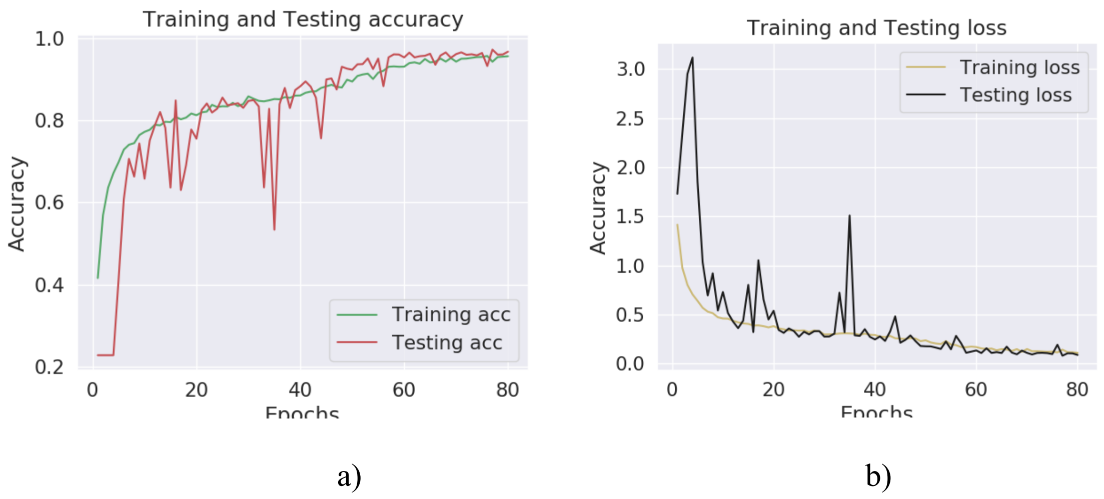
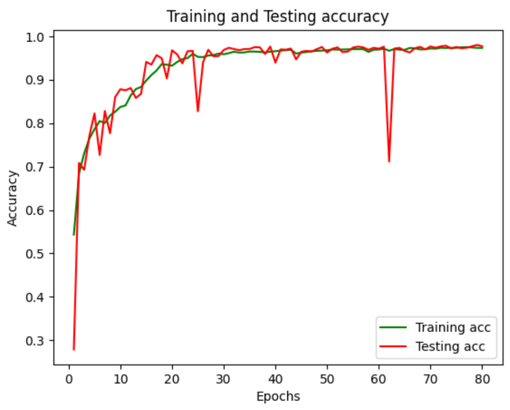
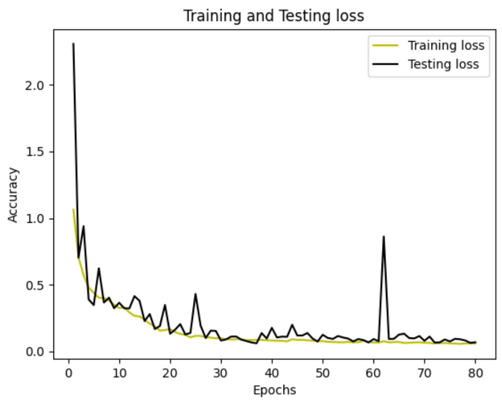


Figure 33. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 128 with 80 epochs, and with pre-processing with a 150x150 image resolution

In Figure 34, experiments shown were run with *lr* of 0.0001 for the batch size of 32 with image resolution of 150x150. In this experiment, accuracy increased, but the testing accuracy curve obtained with the batch size of 32 strictly followed the training curve. Test accuracy of 97.52% is obtained with this experiment but learned very quickly, which is not suitable for new images to train.

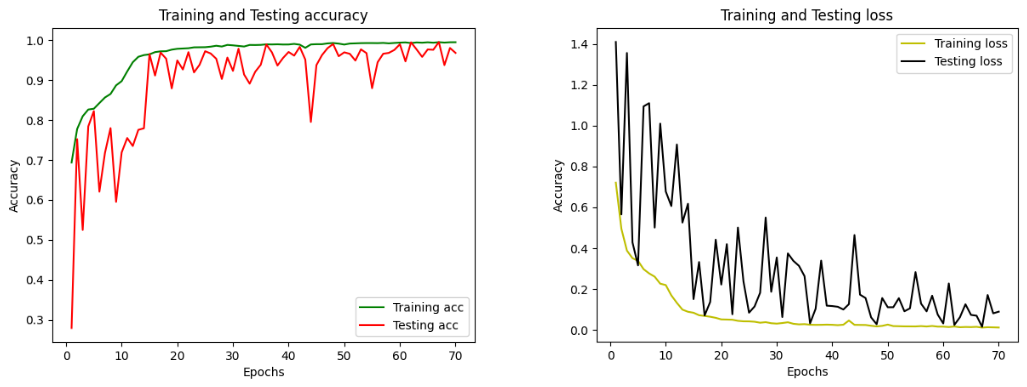




a) b)

Figure 34. a) Accuracy, and b) Loss curve with *lr* 0.0001, for a batch size of 32 with 80 epochs, and with pre-processing included in 150x150 image resolution

Further, more experiments performed on this architecture with *lr* 0.001 for a batch size of 64 with an image resolution of 200x200 shown in Figure 35, and confusion matrix in Figure 36. The testing accuracy of 96.86% was achieved. It shows with each epoch, losses were reduced, and weights were updating which leads to improvement in testing accuracy



a) b)

Figure 35. a) Accuracy b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 70 epochs, and with pre-processing with a 200x200 image resolution

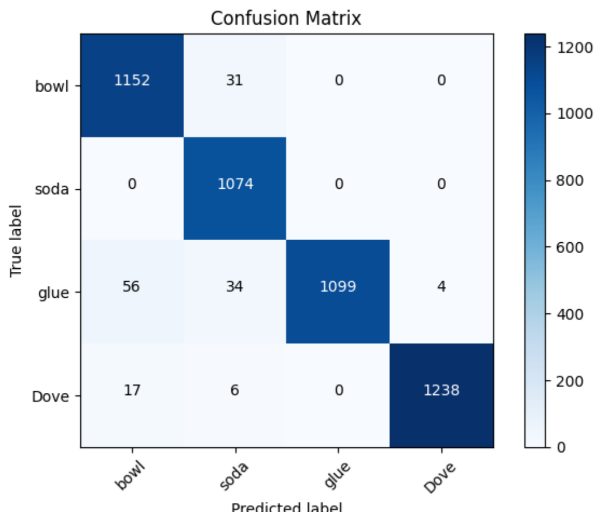
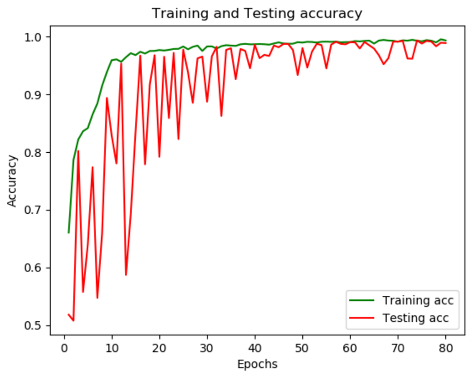
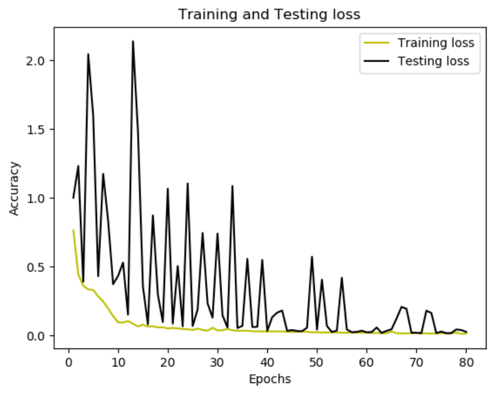


Figure 36. Confusion matrix for architecture II for a *lr* of 0.001, batch size of 64 with 70 epochs, and with an image size of 200x200

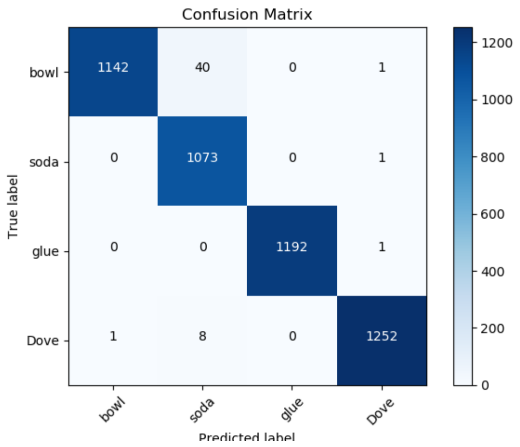
In Figure 37, the architecture was set with *lr* 0.001 for a batch size of 64 with an image resolution of 150x150 to get proficiency, here it shows how the weights updated with each epoch and reducing the error on each epoch. It shows that the testing accuracy curve is following the training curve without overfitting. Confusion matrix in Figure 38 shows very few images for '*Bowl'* misclassified as '*Soda can.*' Overall, the testing accuracy was 98.90%.





a) b)

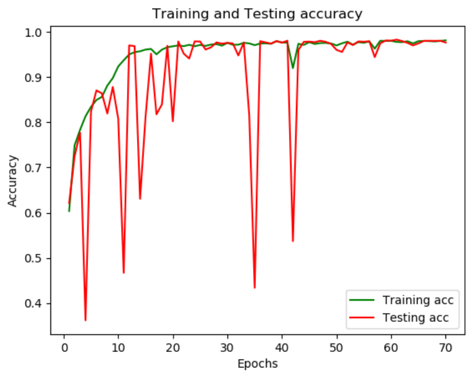
Figure 37. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 80 epochs, and with pre-processing with a 150x150 image resolution


Confusion Matrix
True label
Predicted label

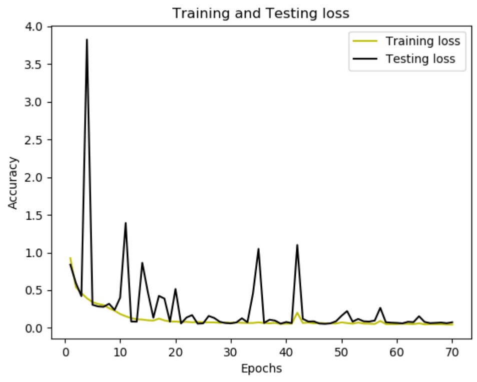
	bowl	soda	glue	Dove
bowl	1142	40	0	1
soda	0	1073	0	1
glue	0	0	1192	1
Dove	1	8	0	1252

Figure 38. Confusion matrix for architecture II for a *lr* of 0.001, batch size of 64 with 80 epochs, and with the image size of 150x150

More rigorous testing followed with 200x200 image size for architecture with a batch size of 32 and 0.0001 *lr* can be seen below in Figure 39, with the smaller batch size of 32, spikes were seen in this experiment as it did not learn all the features, and when difficult test images are presented, it gives less testing accuracy can be seen between epoch 30 to 50, two spikes. Figure 40 confusion matrix show '*Bowl*' images are misclassified as '*Soda*,' more infrequently from the previous hyperparameters.



a)



b)

Figure 39. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 32 with 70 epochs, and with pre-processing with a 200x200 image resolution

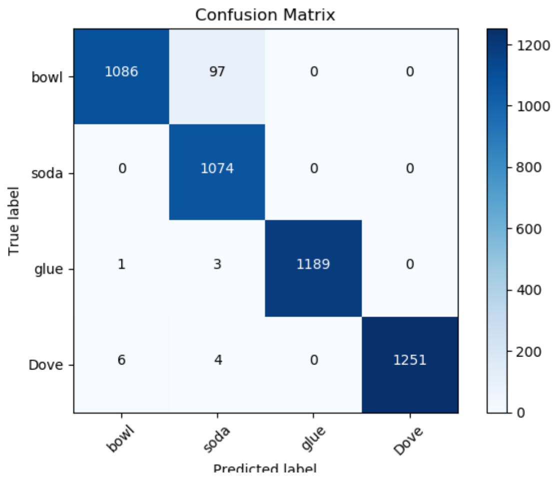
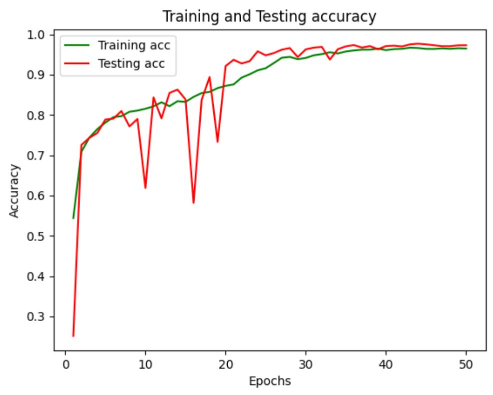
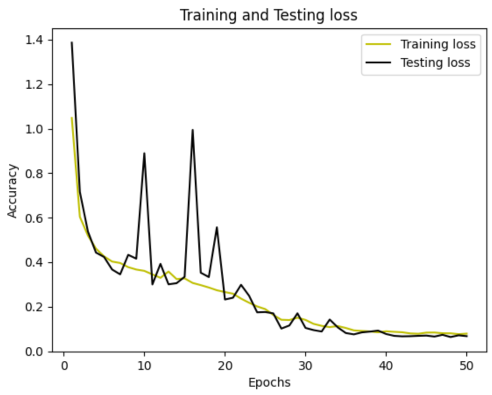


Figure 40. Confusion matrix for architecture II for a *lr* of 0.0001, batch size of 32 with 70 epochs, and with the image size of 200x200

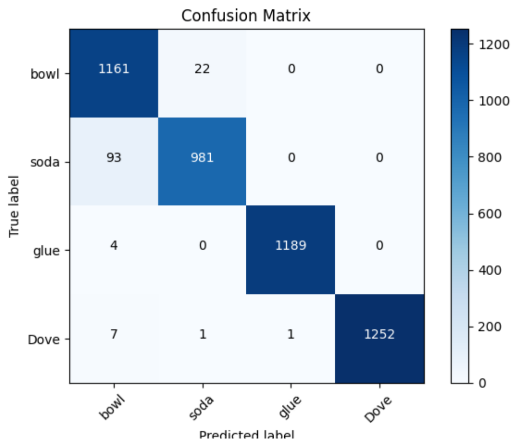
Here, Figure 41 shows the *lr* 0.0001 for a batch size of 64 with 50 epochs for 200x200 images. As shown in the graph, there are some initial hiccups in the learning curves, but after the 30th epoch, it was saturated, and it was not learning new features. Test accuracy obtained by this experiment is 97.28%. Figure 42 shows the confusion matrix with mostly '*Soda can*' images misclassified as '*Bowl*.'





a) b)

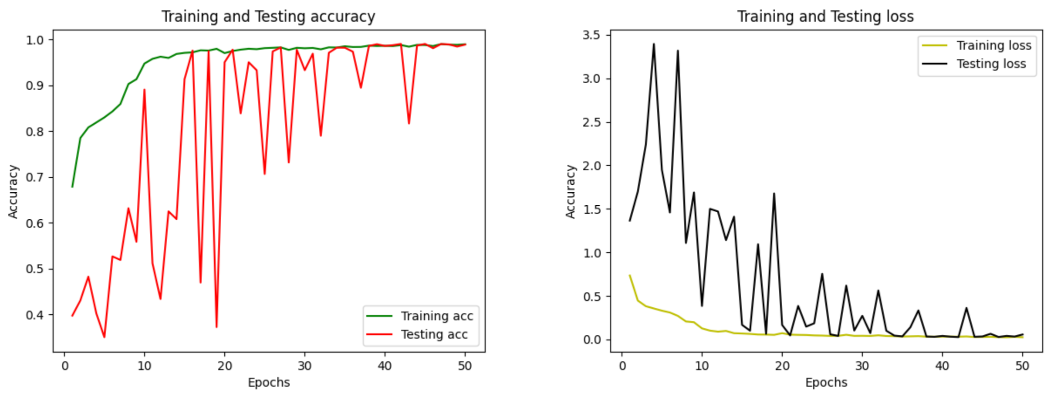
Figure 41. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 64 with 50 epochs, and with pre-processing with a 200x200 image resolution



True label \ Predicted label	bowl	soda	glue	Dove
bowl	1161	22	0	0
soda	93	981	0	0
glue	4	0	1189	0
Dove	7	1	1	1252

Figure 42. Confusion matrix for architecture II for a *lr* of 0.0001, batch size of 64 with 50 epochs, and with the image size of 200x200

Test accuracy obtained for *lr* 0.001 for a batch size of 64 with 50 epochs for 200x200 image resolution by this experiment is 98.83%. Figure 43 shows that the model with less batch size upgrades weight frequently, which leads to variation in the testing curve. This experiment performed better in the classification of objects, and a minimal number of images were misclassified, as seen in confusion matrix in Figure 44.



a) b)

Figure 43. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 50 epochs, and with pre-processing with a 200x200 image resolution

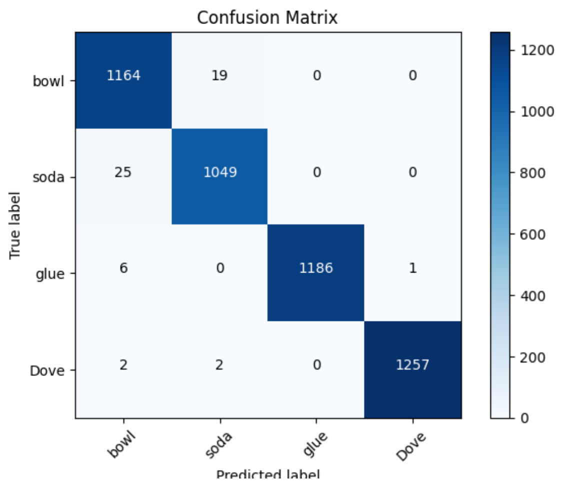


Figure 44. Confusion matrix for architecture II for a *lr* of 0.001, batch size of 64 with 50 epochs, and with the image size of 200x200

This experiment on architecture II performed with a batch size of 128, *lr* of 0.0001 for 50 epochs with 200x200 image size. Figure 45 obtained from this shows a little bumpy because of difficulty predicting testing images for testing accuracy, but performing good in the confusion matrix in Figure 46. Test accuracy of 97.07% and training accuracy of 96.05% is obtained.

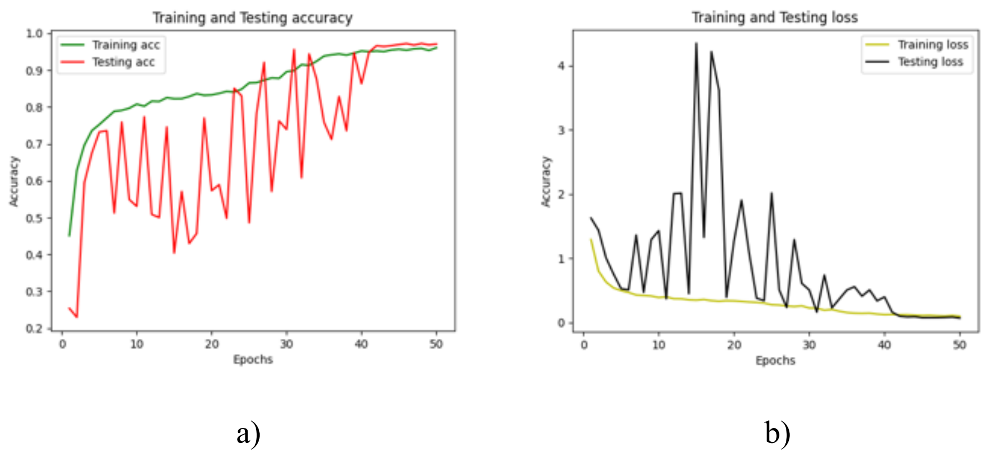
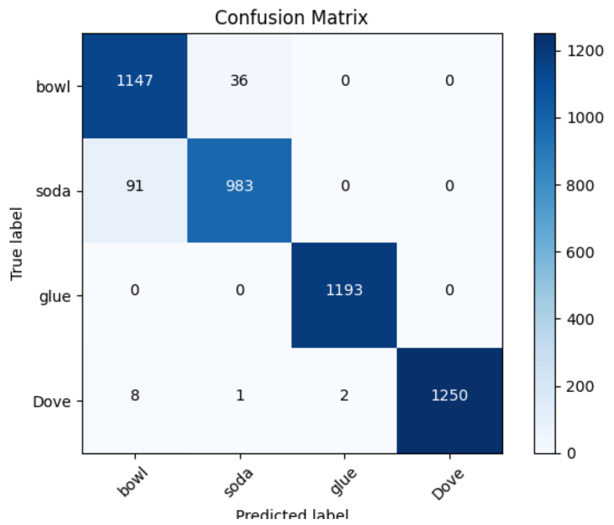


Figure 45. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 128 with 50 epochs, and with pre-processing with a 200x200 image resolution



	bowl	soda	glue	Dove
bowl	1147	36	0	0
soda	91	983	0	0
glue	0	0	1193	0
Dove	8	1	2	1250

Figure 46. Confusion matrix for architecture II for a *lr* of 0.0001, batch size of 128 with 50 epochs, and with the image size of 200x200

More experiments were performed for this architecture to get better insight with the batch size of 128, *lr* of 0.001 for 50 epochs with 200x200 image size. Figure 47 obtained from this shows more stability than the previous one for testing loss and performing better in the confusion matrix in Figure 48. Test accuracy of 98.32% and training accuracy of 98.83% is obtained.

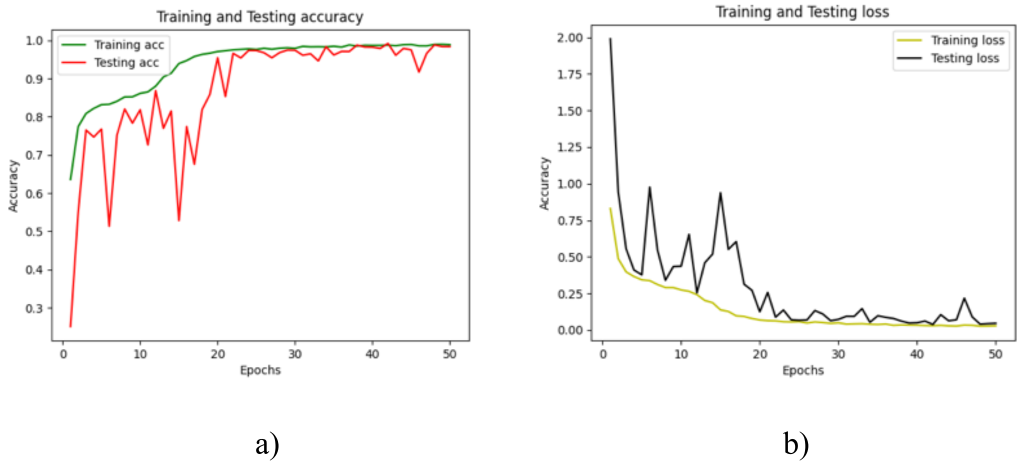


Figure 47. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 128 with 50 epochs, and with pre-processing with a 200x200 image resolution.

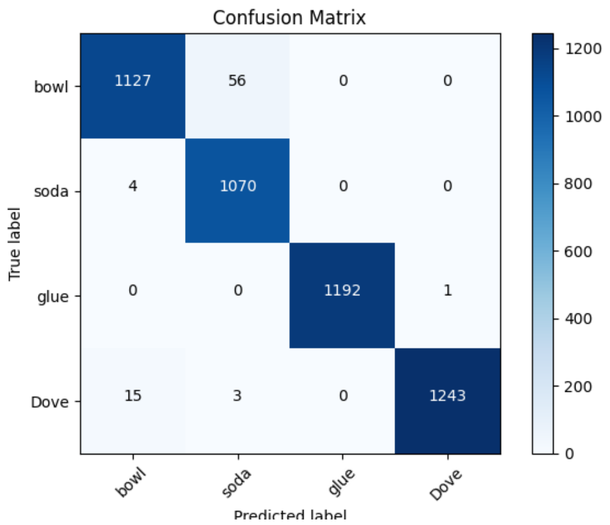
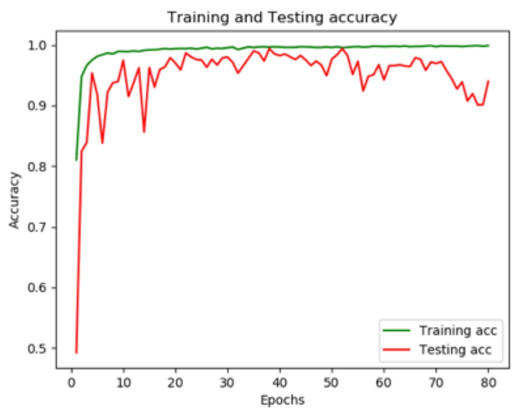


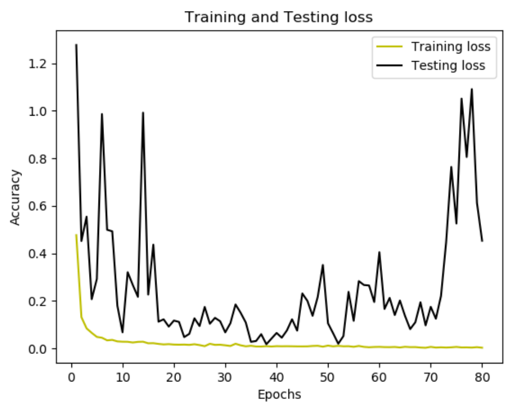
Figure 48. Confusion matrix for architecture II for a *lr* of 0.001, batch size of 128 with 50 epochs, and with the image size of 200x200

Performance of Architecture III

As discussed in the methodology section, architecture III uses different arrangements for the convolution layers, batch normalization layers compared to the other three architectures on testing the dataset for different experiments, including batch size, learning rate, and image resolution optimization. This architecture took fourteen days, sometimes even more, as it has more trainable parameters (in the previous chapter, Table 3). Here, Figure 49 a) the accuracy and b) loss curves results are obtained on running the experiments with 0.001 learning rate for a batch size of 64 for an image size of 150x150. It can be analyzed in Figure 49 that the architecture III learned all the features very quickly. It is not a good thing to have in architecture performance as it does not allow new features to learn from the new images in the future. As it learned quickly, testing losses are high, and there are many variations in the plot.



a)



b)

Figure 49. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 80 epochs, and with the image size of 150x150

Test accuracy obtained by this architecture with these hyperparameters is 93.99%, but when confusion matrix is seen in Figure 50, it can be observed that the model misclassified ‘*Dove bar*’ as a ‘*Bowl.*’

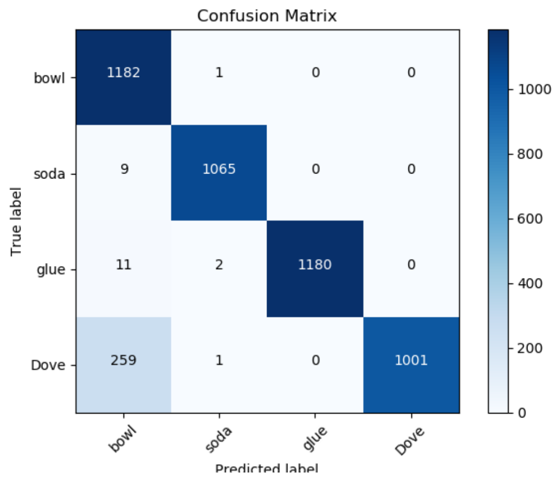
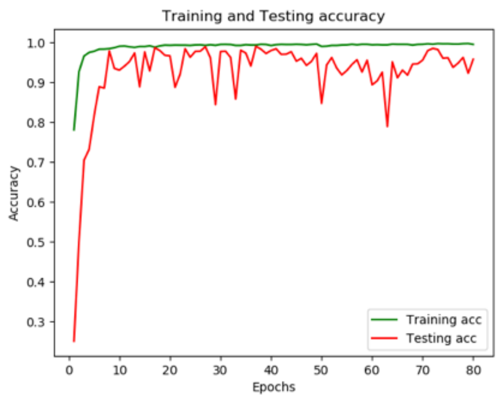
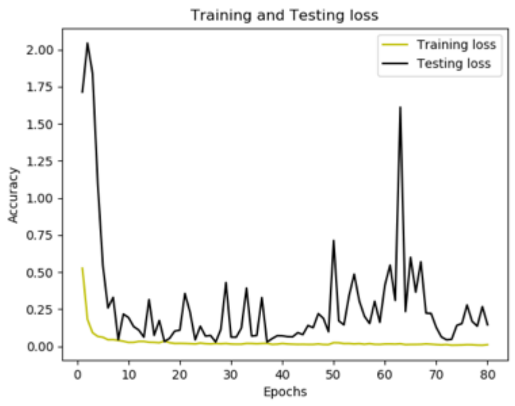


Figure 50. Confusion matrix for architecture III for a *lr* of 0.001, batch size of 64 with 80 epochs, and with the image size of 150x150

Similarly, more experiments were performed in the Figure 51 and confusion matrix shown in Figure 52 with the *lr* 0.001 a batch size of 128 with 80 epochs with pre-processing included in 150x150 image. Test accuracy obtained by the tuning image size was 95.75 %, a 1.76% higher than a batch size of 64. In Figure 51a), the accuracy curve is observed that training curve followed the same path as in Figure 26, but losses are less in this experiment with the increased the batch size to 128. In Figure 52, confusion matrix indicates that 161 ‘*Dove bar*’ were misclassified as ‘*School glue.’*





a) b)

Figure 51. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 128 with 80 epochs, and with the image size of 150x150

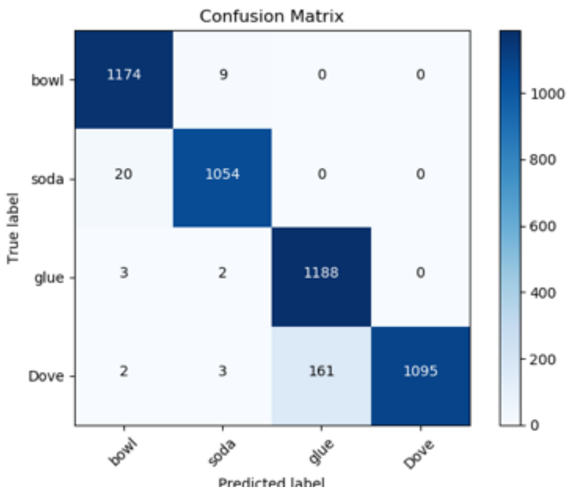
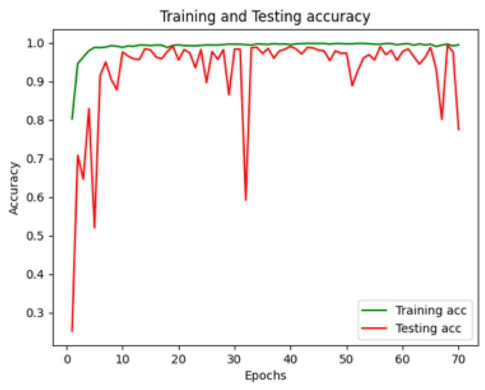
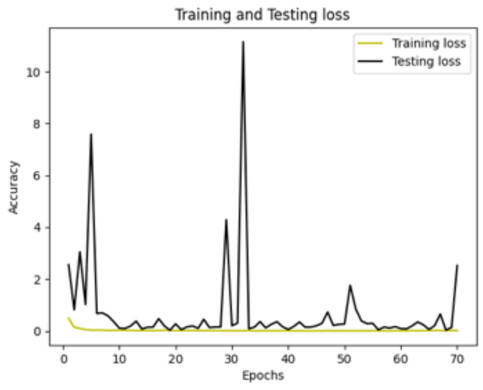


Figure 52. Confusion matrix for architecture III for a *lr* of 0.001, batch size of 128 with 80 epochs, and with the image size of 150x150

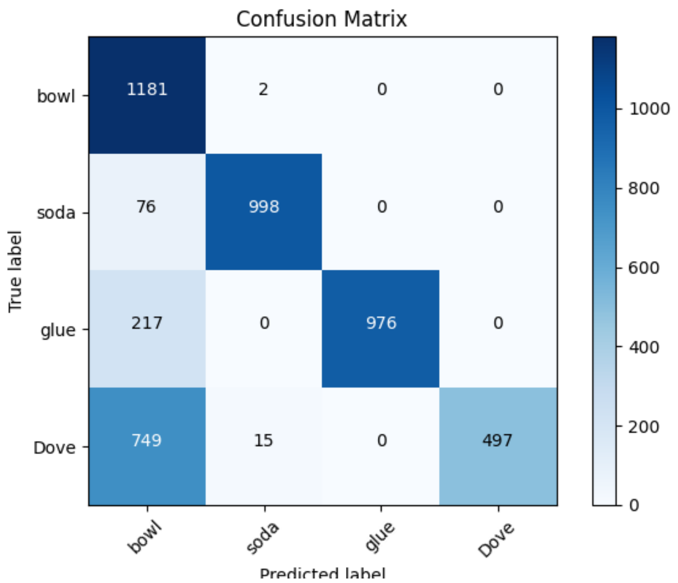
Some more experiments were performed with variation in image resolution to 200x200. In Figure 53 and Figure 54, the testing accuracy obtained was 77.52%. As the image resolution increases, the number of parameters also increases in the model and needs more time to train. It is clearly seen if the model does not learn properly, then it will not predict correctly with a high rate of misclassifications. This architecture took too much time to finish the job with the low learning rate, and most of the time could not finish in the given defined walltime configution set on the LEAP cluster.





a) b)

Figure 53. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 70 epochs, and with pre-processing with a 200x200 image resolution

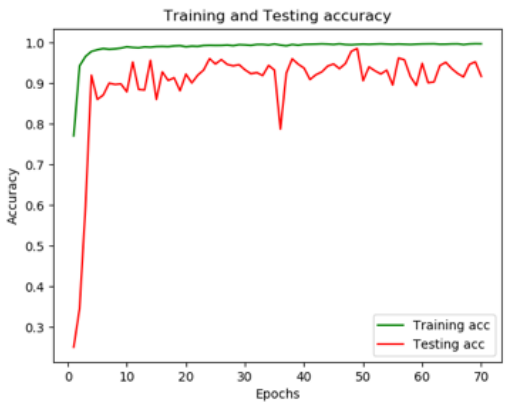


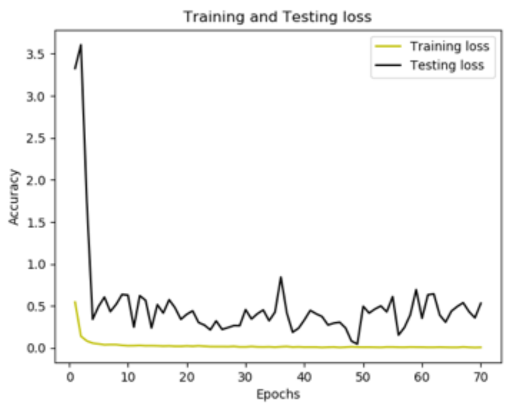
	bowl	soda	glue	Dove
bowl	1181	2	0	0
soda	76	998	0	0
glue	217	0	976	0
Dove	749	15	0	497

Figure 54. Confusion matrix for architecture III for a *lr* of 0.001, batch size of 64 with 70 epochs, and with the image size of 200x200

Performance of Architecture IV

Architecture IV uses six convolution layers with a filter size of 3x3 for the feature extraction. More details have been discussed in the architecture design in the methodology section. Experiments were performed to get the performance of this design. The obtained results are shown below with the different learning rates, batch sizes, and image sizes. In Figure 55 a), the accuracy plot is shown for architecture IV, here six convolution layers were used to train the model with more no linear features, but this architecture learned all the features way early with the additional information from Depth image information it performs well in testing accuracy. Accuracy attained by this architecture is shown in Figure 55 is at 91.74% with the *lr* of 0.001, and a batch size of 64 with 70 epochs.





a) b)

Figure 55. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 70 epochs, and with pre-processing with a 200x200 image resolution

Confusion matrix obtained from this is shown in Figure 56. In the confusion matrix, '*Dove bar*' is misclassified as '*Soda can*,' and '*Soda can*' also misclassified as '*Bowl*'. The F1-Score obtained for the '*Bowl*:' 0.91, for '*Soda Can*:' 0.85, '*School glue*:' 0.91, '*Dove*:' 0.91.

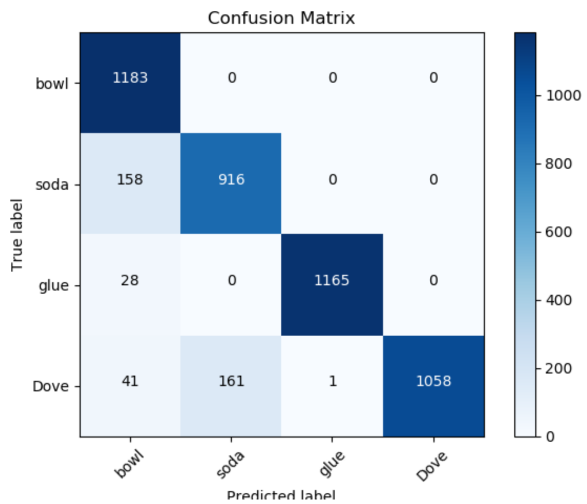
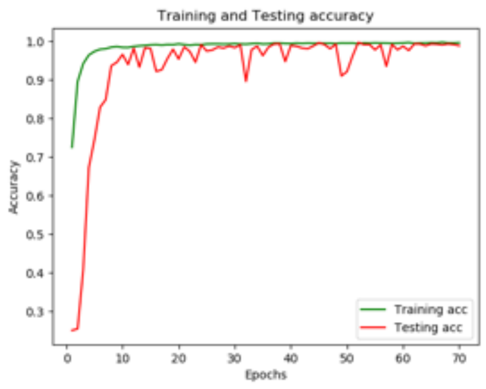
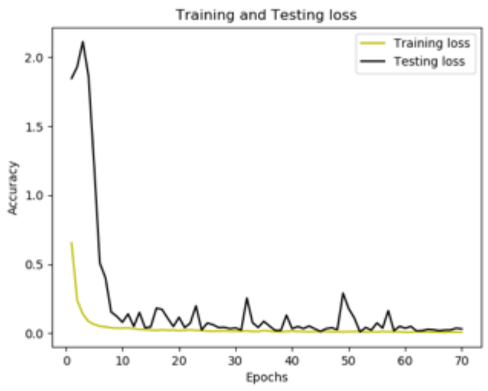


Figure 56. Confusion matrix for architecture IV for a *lr* of 0.001, batch size of 64 with 70 epochs, and with the image size of 200x200

Furthermore, architecture IV investigated all other combinations performed on other architecture to get more depth insight into this model and comparison and confusion matrix. In Figure 57, the experiments were performed with a *lr* of 0.001 for a batch size of 128 with 70 epochs for an image size of 200x200, it includes all the pre-processing for the brightness and contrast, but in training accuracy curve it learned all the features in the first ten epochs which is not good as it does not allow more features to learn. Here, classification accuracy increased in Figure 58, but the learning curve reached saturation way earlier.



a)



b)

Figure 57. a) Accuracy, and b) Loss curve with *lr* 0.001, for a batch size of 128 with 70 epochs, and with pre-processing included in 200x200 image resolution

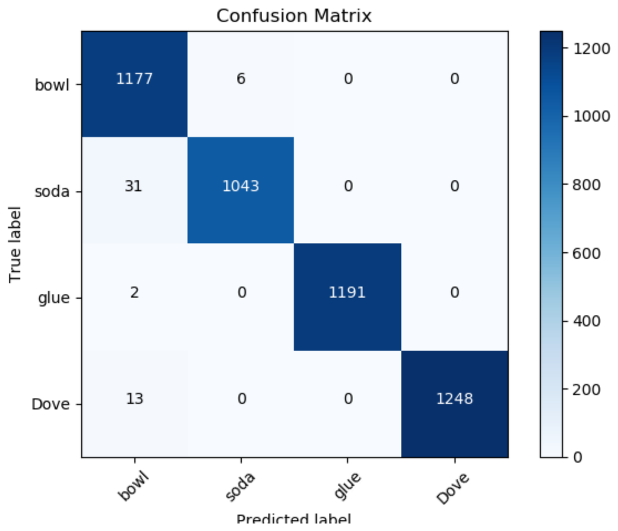
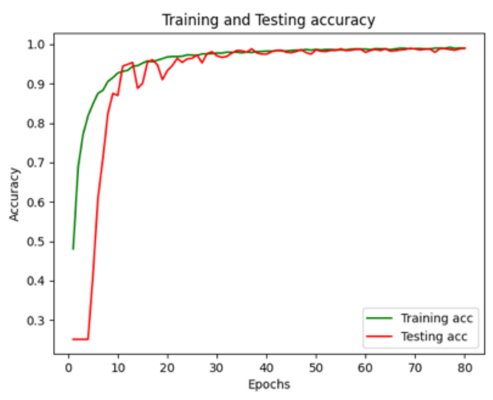
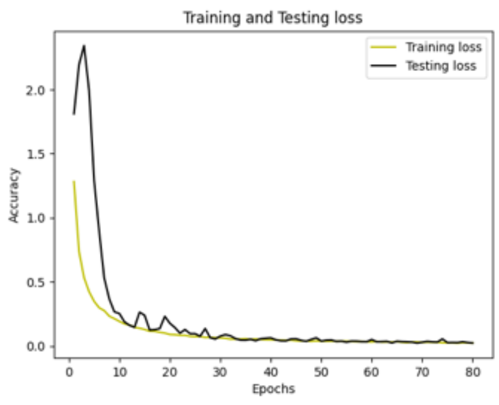


Figure 58. Confusion matrix for architecture IV for a *lr* of 0.001, batch size of 128 with 70 epochs, and with the image size of 200x200

Experiments were performed; results are shown in Figures 59 and 60, but no improvements were observed to overcome this problem. Architecture IV had fewer parameters than architecture I and architecture III; therefore, it took less time to train. Overall, this architecture tested thoroughly with multiple changes to tune it properly but did not generalize the dataset to acceptable levels of accuracy.

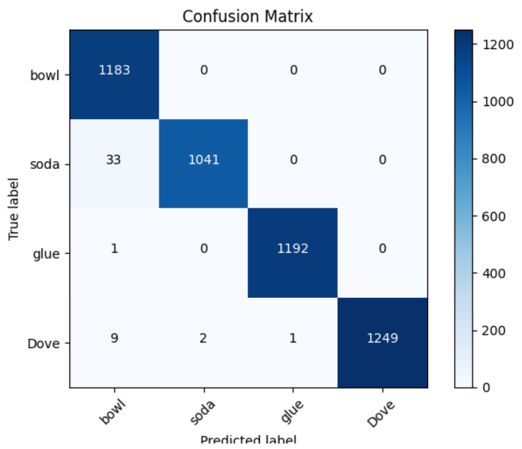


a)



b)

Figure 59. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 128 with 80 epochs, and with pre-processing included in 150x150 image resolution

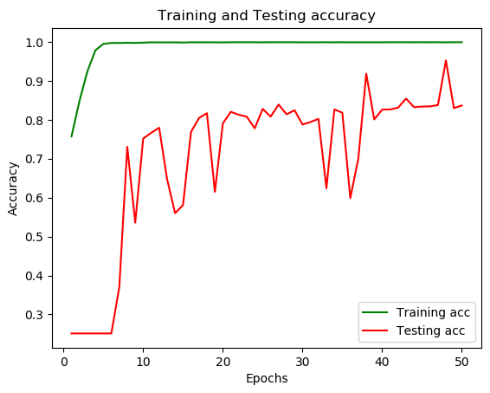


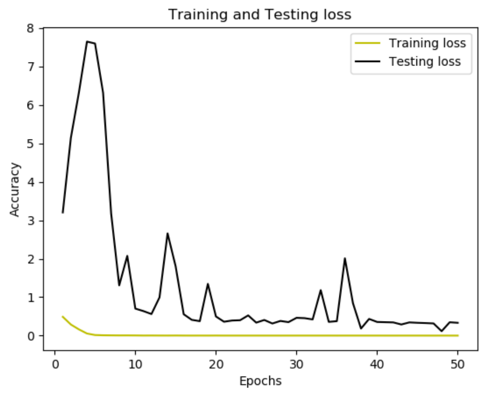
	bowl	soda	glue	Dove
bowl	1183	0	0	0
soda	33	1041	0	0
glue	1	0	1192	0
Dove	9	2	1	1249

Figure 60. Confusion matrix for architecture IV for a *lr* of 0.0001, batch size of 128 with 80 epochs, and with the image size of 150 x 150

For Dark images with architecture II

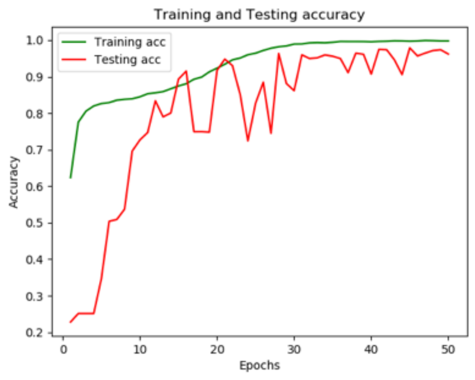
Moving Further, after completion of rigorous testing of all the architectures and gaining an idea about the performance, architecture II trained and tested again on the only dark images to see the performance with only low illumination. n Figure 61 a) was observed that the testing accuracy increased with the increase in epochs. For a *lr* of 0.001 with a batch size of 128, it achieved an accuracy of 83.72%. However, on increasing the *lr* to 0.0001, batch size of 128 for an image of 200x200, the accuracy improved to 96.18%, and learning curves also improved, as shown in Figure 62.

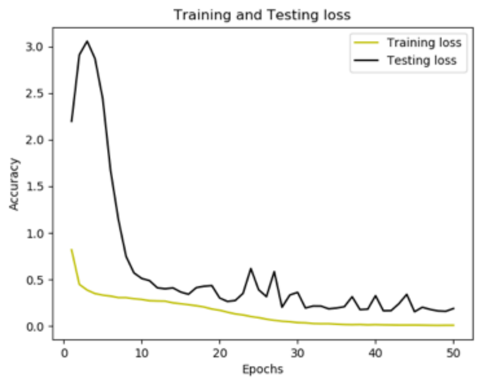




a) b)

Figure 61. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 128 with 50 epochs, and with pre-processing with a 200 x 200 image resolution

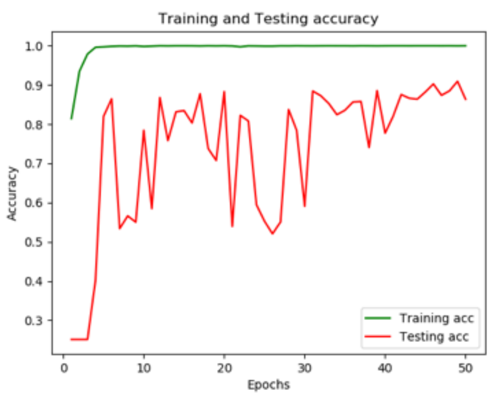


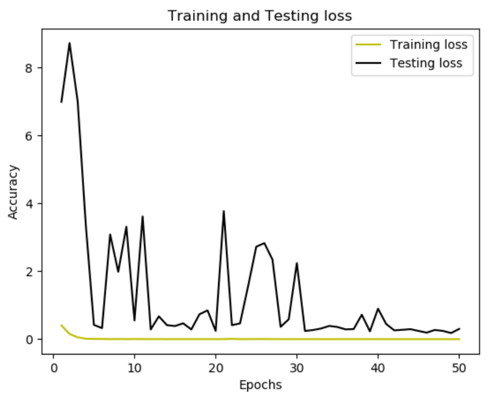


a) b)

Figure 62. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 128 with 50 epochs, and with pre-processing with a 200 x 200 image resolution

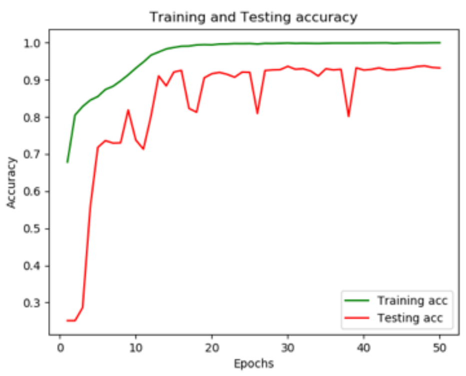
A batch size changed to 64 to gain more architecture II performance expertise to perform more architecture II experiments. The accuracy of batch 64 with 0.001 and 0.0001 is 86.37% and 93.19%. It can be seen in Figure 63 with the testing accuracy curve, there was variation in testing performance with overfitting in the accuracy curves and did not make any improvement, although it performed better in Figure 64 with 93.19% accuracy but not better than the previous experiment with a batch size 128.

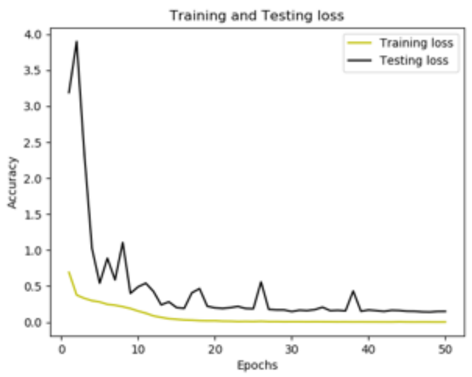




a) b)

Figure 63. a) Accuracy, and b) Loss curve with a *lr* of 0.001, for a batch size of 64 with 50 epochs, and with pre-processing with a 200 x 200 image resolution





a) b)

Figure 64. a) Accuracy, and b) Loss curve with a *lr* of 0.0001, for a batch size of 64 with 50 epochs, and with pre-processing with a 200 x 200 image resolution

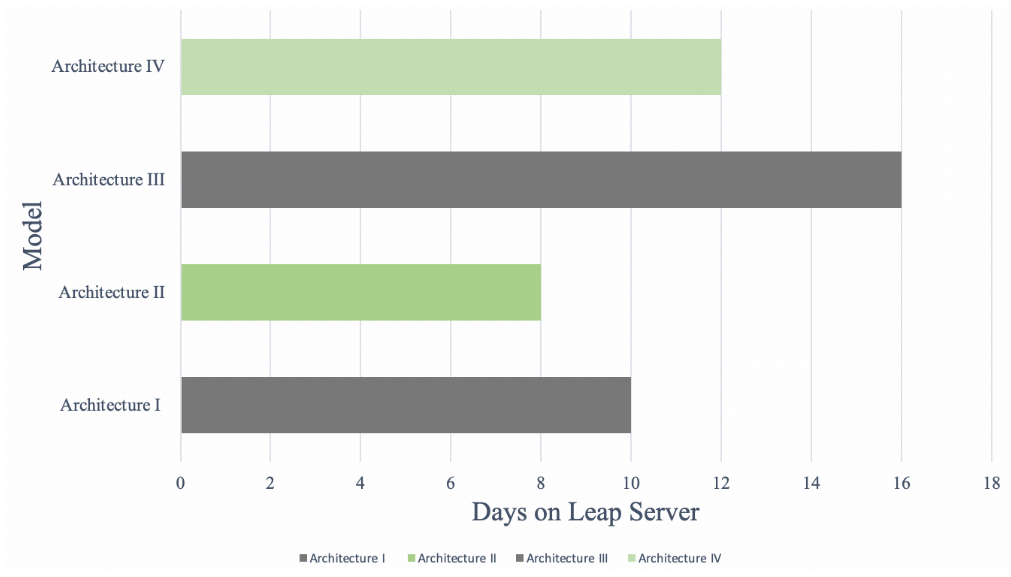


Figure 65. Comparison of time taken by different architectures to train on LEAP server in days

Figure 65 results from the LEAP server for different architectures training and testing with the Princeton and Washington dataset. In this plot, the y-axis models are shown, and on the x-axis number of days are mentioned. Here, it is noticeably seen that the least number of days were taken by architecture II. It is shown in green color as it also performed better in classification accuracy. Architecture I, shown in dark gray color, from the above results, it is seen that it did not perform to the expectation for this task as compared to any other results. It took a smaller number of days than architecture IV and architecture III, but the learning curves and overall performance was not good compared to other architectures. Architecture III took more than fourteen days and could not complete the training and evaluation within the wall time restriction on the cluster environment. Architecture IV, also shown in green color, it had a more significant number of convolutional layers, but because of other parameters, it took fewer days than architecture III and performed somewhat better than architecture I and III.

The designed architecture was also tested after saving the trained model with the images taken by RGB-D Intel RealSense camera in the different background for the '*Bowl*,' '*Soda can*,' '*School glue*,' '*Dove bar*.' The images taken by the camera give 57% accuracy, which is less because the images are different from the training images. Confusion matrix are shown in Figure 66 for image prediction.

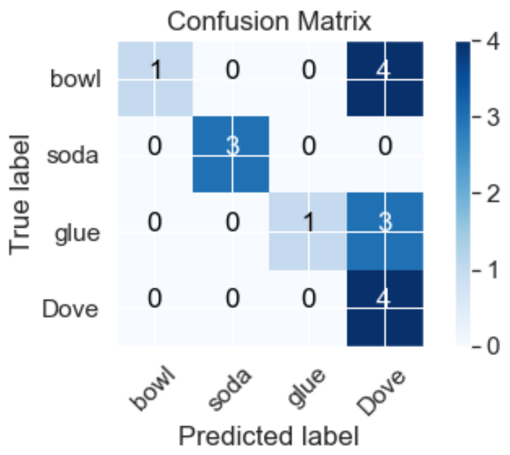


Figure 66. Confusion matrix for the prediction with the images taken by the camera.

In this chapter, Experiments performed with architectures I, II, III, and IV are discussed. Different hyperparameters of architectures were changed in all the experiments, such as batch size, epochs, learning rate, and performance was analyzed and discussed. Architecture I was learning too fast; after changing different hyperparameters, it did not perform well on learning curves. Architecture II performed well in speed and accuracy; the best result obtained was 98.90% with a batch size of 64 and with 0.001 as the learning rate. Architecture III did not perform well on learning curves and accuracy, and it took many days to train with a huge number of parameters. Architecture IV performed better than architecture I and III but not better than architecture II on learning curves and metrics.

VIII. CONCLUSION

In this thesis, the objective was to design a deep neural network for object recognition capabilities in the warehouse automation for a pick and place robot arm with a vision system. A thorough literature review was performed for the thesis work and to design of architecture. The design approach was to obtain an optimized architecture model solution to overcome the problem of low illumination, different lighting conditions, and partial images of the objects. Four Deep Convolutional Neural Network (DCNN) were designed and investigated in-depth with the different parameters. The Princeton and Washington datasets were used to develop, train, and test the DCNN models. The Princeton and Washington datasets had 39 and 51 different classes of objects images, but only four objects images were considered for this work: '*Bowl*,' '*Soda can*,' '*School glue*,' and '*Dove bar*.'

The DCNN models have two streams consisting of one for the RGB image input, and the second one for the Depth image input. For training of the designed DCNN architectures, the datasets were pre-processed by using OpenCV for the different illumination conditions like brightness, contrast, and resize of image resolution before passing through the first convolution layers of each architecture. After preprocessing the dataset, the RGB and Depth images are passed to their respective network stream of the DCNN model. In each stream, different types of layers were used; convolution layer, maxpool layer, batch normalization, dropout, activation layer and dense layer to learn the features of the RGB and Depth input images. After passing through two different streams of RGB and Depth networks, these two networks were combined using the concatenation layer to combine information from both the streams. Finally, the Softmax layer was used as the output to provide a predicted percentage values of the multiclass classification. Four architectures were designed and tested for the mentioned datasets. In all of the four architectures design, experiments were performed to tune the hyperparameters, such as the batch size, learning rate, image resolution, epochs, dropout rates, and the number of filters.

Finally, all four architectures were tested on the LEAP cluster and in a single HiPE server. Testing of all the models, architecture II performed well compared to other architectures in classification accuracy, loss rate learning curves, and confusion matrix metrics for training and testing dataset. Apart from performing well for all the obtained evaluation metrics, it was computationally efficient during training in comparison to the other architectures on the LEAP cluster. The best testing accuracy achieved by architecture II was 98.90%.

IX. FUTURE WORK

For future work, robots and AI will change how people work, affecting industries, with more robots than engineers. More exact vision is required with better precision to make a robot reliable and innocuous. The automation base model will be added to the SSD detection network and actualized to the mechanical robot after thorough testing in a current situation, which can get higher accuracy for incomplete pictures. More objects can be added to the quality of preparing with a bigger dataset. For the current grasping detection, separate grasping planning is required, which can be improved and combined with the same model. Other techniques can also be implemented for training robots using reinforcement learning to make it more robust.

Furthermore, the designed model uses five convolution layers and different layers for RGB-D images in each network stream, which can also be improved to increase the accuracy. Along with this, the addition of more objects to the detection network with more challenges such as different/irregular shapes of objects and adverse lighting conditions would make it even stronger. For irregular and soft objects, a suction gripper can be used, and a pressure sensor can also be used with a camera to detect pressure required to grasp.

REFERENCES

[1]. P. Sharma and D. Valles, "Deep Convolutional Neural Network Design Approach for 3D Object Detection for Robotic Grasping," *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2020, pp. 0311-0316, doi: 10.1109/CCWC47524.2020.9031186.

[2]. A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla and J. Azorin-Lopez, "PointNet: A 3D Convolutional Neural Network for real-time object class recognition," 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, 2016, pp. 1578-1584, doi: 10.1109/IJCNN.2016.7727386.

[3]. S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, 2017, pp. 769-776, doi: 10.1109/IROS.2017.8202237.

[4]. SUN RGB-D: An RGB-D Scene Understanding Benchmark Suite, http://rgbd.cs.princeton.edu, [Accessed: on Oct. 27, 2019].

[5]. RGB-D Object Dataset, https://rgbd-dataset.cs.washington.edu [Accessed: Oct. 27, 2019].

[6]. A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller and W. Burgard, "Multimodal deep learning for robust RGB-D object recognition," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, 2015, pp. 681-687, doi: 10.1109/IROS.2015.7353446.

[7]. M. B. Soares and S. Wermter, "Point Cloud Object Recognition using 3D Convolutional Neural Networks," 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489270.

[8]. R. Jonschkowski, C. Eppner, S. Höfer, R. Martín-Martín and O. Brock, "Probabilistic multi-class segmentation for the Amazon Picking Challenge," 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, 2016, pp. 1-7, doi: 10.1109/IROS.2016.7758087.

[9]. A. Zeng et al., "Multi-view self-supervised deep learning for 6D pose estimation in the Amazon Picking Challenge," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 1386-1383, doi: 10.1109/ICRA.2017.7989165

[10]. A. P. del Pobil et al., "UJI RobInLab's approach to the Amazon Robotics Challenge 2017," 2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Daegu, 2017, pp. 318-323, doi: 10.1109/MFI.2017.8170448.

[11]. C. H. Corbato, M. Bharatheesha, J. van Egmond, J. Ju and M. Wisse, "Integrating Different Levels of Automation: Lessons From Winning the Amazon Robotics Challenge 2016," in IEEE Transactions on Industrial Informatics, vol. 14, no. 11, pp. 4916-4926, Nov. 2018, doi: 10.1109/TII.2018.2800744.

[12]. M. Schwarz et al., "NimbRo picking: Versatile part handling for warehouse automation," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 3032-3039, doi: 10.1109/ICRA.2017.7989348.

[13]. C. Rennie, R. Shome, K. E. Bekris and A. F. De Souza, "A Dataset for Improved RGBD-Based Object Detection and Pose Estimation for Warehouse Pick-and-Place," in IEEE Robotics and Automation Letters, vol. 1, no. 2, pp. 1179-1185, July 2016, doi: 10.1109/LRA.2016.2532924.

[14]. X. Lu, X. Kang, S. Nishide and F. Ren, "Object detection based on SSD-ResNet," *2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Singapore, 2019, pp. 89-92, doi: 10.1109/CCIS48116.2019.9073753.

[15]. A. Womg, M. J. Shafiee, F. Li and B. Chwyl, "Tiny SSD: A Tiny Single-Shot Detection Deep Convolutional Neural Network for Real-Time Embedded Object Detection," 2018 15th Conference on Computer and Robot Vision (CRV), Toronto, ON, 2018, pp. 95-101, doi: 10.1109/CRV.2018.00023.

[16]. S. Hsiao, J. Zhan and C. Lin, "Low-Complexity Deep Neural Networks for Image Object Classification and Detection," 2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Bangkok, Thailand, 2019, pp. 313-316, doi: 10.1109/APCCAS47518.2019.8953165.

[17]. Chengcheng Ning, Huajun Zhou, Yan Song and Jinhui Tang, "Inception Single Shot MultiBox Detector for object detection," 2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), Hong Kong, 2017, pp. 549-554, doi: 10.1109/ICMEW.2017.8026312.

[18]. W. Liu, D. Anguelov, D. Erhan, and C. Szegedy, “SSD: Single Shot MultiBox Detector,” ECCV, vol. 1, pp. 21–37, 2016, doi: 10.1007/978- 3-319-46448-0.

[19]. Z. Dongtao, C. Jie, Y. Xing, S. Hui and S. Liangliang, "Traffic Sign Detection Method of Improved SSD Based on Deep Learning," *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2018, pp. 1516-1520, doi: 10.1109/CompComm.2018.8780999.

[20]. A. S. Periyasamy, M. Schwarz and S. Behnke, "Robust 6D Object Pose Estimation in Cluttered Scenes Using Semantic Segmentation and Pose Regression Networks," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, 2018, pp. 6660-6666, doi: 10.1109/IROS.2018.8594406.

[21]. H. Jo, C. Min and J. Song, "Bin Picking System using Object Recognition based on Automated Synthetic Dataset Generation," 2018 15th International Conference on Ubiquitous Robots (UR), Honolulu, HI, 2018, pp. 886-890, doi: 10.1109/URAI.2018.8441811.

[22]. K. Kim, S. Kang, J. Kim, J. Lee, J. Kim and J. Kim, "Multiple objects recognition for industrial robot applications," 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, 2013, pp. 257-259, doi: 10.1109/URAI.2013.6677361.

[23]. R. S. Andersen, C. Schou, J. S. Damgaard and O. Madsen, "Using a Flexible Skill-Based Approach to Recognize Objects in Industrial Scenarios," Proceedings of ISR 2016: 47st International Symposium on Robotics, Munich, Germany, 2016, pp. 1-8.

[24]. K. Kim, J. Cho, J. Pyo, S. Kang and J. Kim, "Dynamic Object Recognition Using Precise Location Detection and ANN for Robot Manipulator," 2017 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), Prague, 2017, pp. 237-241, doi: 10.1109/ICCAIRO.2017.52.

[25]. Xi Chen, Jan Guhl, "Industrial Robot Control with Object Recognition based on Deep Learning," 7th CIRP Conference on Assembly Technologies and Systems, Procedia CIRP 76 (2018) 149-154.

[26]. Aurélien Géron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition", Publisher: O'Reilly Media, Inc., Release Date: September 2019, ISBN: 9781492032649

[27]. A. Rosebrock, Deep Learning for Computer Vision with Python, 1.3.0. PyImageSearch.com, 2018.

[28]. Sebastian Raschka, https://sebastianraschka.com/ [Accessed: Oct. 6th,2020]

[29]. Geoffrey E. Hinton et al., "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors," arXiv preprint arXiv:1207.0580 (2012).

[30]. Nitish Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research* 15 (2014): 1929–1958

[31]. R. Girshick, "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.

[32]. S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, June 2017, doi: 10.1109/TPAMI.2016.2577031.

[33]. P. Adarsh, P. Rathi and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2020, pp. 687-694, doi: 10.1109/ICACCS48705.2020.9074315.

[34]. D. Temel, J. Lee and G. Alregib, "CURE-OR: Challenging Unreal and Real Environments for Object Recognition," *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Orlando, FL, 2018, pp. 137-144, doi: 10.1109/ICMLA.2018.00028.

[35]. Griffiths, David, and Jan Boehm. "A Review on Deep Learning Techniques for 3D Sensed Data Classification." *Remote Sensing* 11.12 (2019): 1499. Crossref. Web.

[36]. Wikipedia, www.wikipedia.org [Accessed: Oct. 28,2019]

[37]. High-Performance Engineering (HiPE) Research Group, https://hipe.wp.txstate.edu/technology/ [Accessed: Aug. 28, 2020].

[38]. S. Zhai, D. Shang, S. Wang and S. Dong, "DF-SSD: An Improved SSD Object Detection Algorithm Based on DenseNet and Feature Fusion," in *IEEE Access*, vol. 8, pp. 24344-24357, 2020, doi: 10.1109/ACCESS.2020.2971026.

[39]. Shehan Caldera, Alexander Rassau and Douglas Chai, "Review of Deep Learning Methods in Robotic Grasp Detection," *Multimodal Technologies and Interact*. 2018, 2, 57, www.mdpi.com/journal/remotesensing, [Accessed : Oct. 20, 2019]

[40]. W. Czajewski, K. Kołomyjec, "3D-OBJECT DETECTION AND RECOGNITION FOR ROBOTIC GRASPING BASED ON RGB-D IMAGES AND GLOBAL FEATURES," *FOUNDATIONS OF COMPUTING AND DECISION SCIENCES*, Vol. 42, 2017

[41]. Y. Chang, C. G. Li and Y. Hong, "Real-Time Object Coordinate Detection and Manipulator Control Using Rigidly Trained Convolutional Neural Networks," 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), Vancouver, BC, Canada, 2019, pp. 1347-1352, doi: 10.1109/COASE.2019.8842973.

[42]. J. Liu, H. Chen and J. Li, "Faster 3D Object Detection in RGB-D Image Using 3D Selective Search and Object Pruning," 2018 Chinese Control And Decision Conference (CCDC), Shenyang, 2018, pp. 4862-4866, doi: 10.1109/CCDC.2018.8407973.

[43]. Hao Sun, Zehui Meng, Xinxin Du and Marcelo H. Ang Jr., "A 3D-Convolutional Neural NetworkTowards Real-time Amodal 3D-Object Detection," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Oct. 2018, Madrid, Spain

[44]. S. Gupta, P. Arbeláez, R. Girshick and J. Malik, "Aligning 3D models to RGB-D images of cluttered scenes," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 4731-4740, doi: 10.1109/CVPR.2015.7299105.

[45]. J. Lahoud and B. Ghanem, "2D-Driven 3D Object Detection in RGB-D Images," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 4632-4640, doi: 10.1109/ICCV.2017.495.

[46]. J. Kim, "Object detection using RGBD data for interactive robotic manipulation," 2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Kuala Lumpur, 2014, pp. 339-343, doi: 10.1109/URAI.2014.7057537.

[47]. Xinhang Song Luis Herranz, Shuqiang Jiang, "Depth CNNs for RGB-D scene recognition: learning from scratch better than transferring from RGB-CNNs," *Key Laboratory of Intelligent Information Processing of Chinese Academy of Sciences (CAS),* Jan. 2018, China

[48]. Y. Guo, M. Bennamoun, F. Sohel, M. Lu and J. Wan, "3D Object Recognition in Cluttered Scenes with Local Surface Features: A Survey," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 11, pp. 2270-2287, 1 Nov. 2014, doi: 10.1109/TPAMI.2014.2316828.

[49]. N. Correll et al., "Analysis and Observations From the First Amazon Picking Challenge," in IEEE Transactions on Automation Science and Engineering, vol. 15, no. 1, pp. 172-188, Jan. 2018, doi: 10.1109/TASE.2016.2600527.

[50]. E. Martinez-Martin and A. P. del Pobil, "Object Detection and Recognition for Assistive Robots: Experimentation and Implementation," in IEEE Robotics & Automation Magazine, vol. 24, no. 3, pp. 123-138, Sept. 2017, doi: 10.1109/MRA.2016.2615329.

[51]. H. Shin, H. Hwang, H. Yoon and S. Lee, "Integration of deep learning-based object recognition and robot manipulator for grasping objects," 2019 16th International Conference on Ubiquitous Robots (UR), Jeju, Korea (South), 2019, pp. 174-178, doi: 10.1109/URAI.2019.8768650.

[52]. J. M. Cho and K. Kim, "Precise object detection using local feature for robot manipulator," 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, 2017, pp. 497-499, doi: 10.1109/URAI.2017.7992787.

[53]. D. N. Thang et al., "Deep Learning-based Multiple Objects Detection and Tracking System for Socially Aware Mobile Robot Navigation Framework," 2018 5th NAFOSTED Conference on Information and Computer Science (NICS), Ho Chi Minh City, 2018, pp. 436-441, doi: 10.1109/NICS.2018.8606878.

[54]. H. Sun, Z. Meng, P. Y. Tao and M. H. Ang, "Scene Recognition and Object Detection in a Unified Convolutional Neural Network on a Mobile Manipulator," 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, 2018, pp. 5875-5881, doi: 10.1109/ICRA.2018.8460535.

[55]. X. Tang, Y. Song and Y. Zhang, "Feature Fusion for Weakly Supervised Object Localization," 2018 Chinese Automation Congress (CAC), Xi'an, China, 2018, pp. 2548-2553, doi: 10.1109/CAC.2018.8623227.

[56]. J. Yu, K. Weng, G. Liang and G. Xie, "A vision-based robotic grasping system using deep learning for 3D object recognition and pose estimation," 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, 2013, pp. 1175-1180, doi: 10.1109/ROBIO.2013.6739623.

[57]. Y. Zhang, H. Wang and F. Xu, "Object detection and recognition of intelligent service robot based on deep learning," 2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), Ningbo, 2017, pp. 171-176, doi: 10.1109/ICCIS.2017.8274769.

[58]. E. Etemad and Q. Gao, "Object localization by optimizing convolutional neural network detection score using generic edge features," 2017 IEEE International Conference on Image Processing (ICIP), Beijing, 2017, pp. 675-679, doi: 10.1109/ICIP.2017.8296366.

[59]. Jon Krohn, Grant Beyleveld, Aglaé Bassens, “Deep Learning Illustrated: A Visual, Interactive Guide to Artificial Intelligence”, Publisher: Addison-Wesley Professional, Release Date: September 2019, ISBN: 9780135116821