Indoor Navigation Efficiency Improvement in Intelligent Assistive Systems (IAS) Using Neural Networks

by

Amlan Basu
(*Registration Number: 201765106)*

Under the guidance of

Dr. Lykourgos Petropoulakis
(*Primary Supervisor*)
&
Dr. Gaetano Di Caterina
(*Second Supervisor*)

Neuromorphic Lab for AI and Deep Learning Systems
Centre for Signal and Image Processing (CeSIP)
Department of Electronic and Electrical Engineering
University of Strathclyde, Glasgow G1 1XQ, U.K.

This thesis is submitted for the award of the degree of

Doctor of Philosophy

2022

Gitanjali, Poem 35

Where the mind is without fear and the head is held high;
Where knowledge is free;
Where the world has not been broken up into fragments by narrow domestic walls;
Where words come out from the depth of truth;
Where tireless striving stretches its arms towards perfection;
Where the clear stream of reason has not lost its way into the dreary desert sand of dead habit;
Where the mind is led forward by thee into ever-widening thought and action –
Into that heaven of freedom, my Father, let my country awake.

~ Rabindranath Tagore (Nobel Prize, 1913)

Declaration

This thesis is the result of author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Amlan Basu

2022

Acknowledgments

Doctor of Philosophy (Ph.D.) is a degree that many people dream of, but very few get the opportunity to pursue and complete it. I have been one of those fortunate persons who got this opportunity. Ph.D. of any individual is always intriguing because of its nature of sinuousness. The same remained in my case as well. My experience being a Ph.D. student is ineffable. In this anfractuous ineffable journey, there have been many people who continuously espoused me at every moment. Acknowledging or thanking them will never be enough to describe their contributions. Still, I would like to take this opportunity to acknowledge as that would somewhat give me a sense of satisfaction.

First and foremost, I would like to show my highest level of obeisance to my primary supervisor, Dr. Lykourgos Petropoulakis. He is the person who accepted my application for Ph.D. and showed his trust and confidence in me. Even during the journey of my Ph.D., he stood by my side whenever I needed him. Sir always kept a close eye on whatever I was doing and used to meet at least once every week to know where I was in my work. He did the same during the pandemic lockdown through Skype.

Further, how can I forget his meticulous observations that he made on my work and especially the way I presented my work in writing. Being a supervisor, he bothered about what I did, but he also tried to provide many inputs that can help to achieve the aims and objectives. Dr. Petropoulakis, apart from being professional, has always been an amicable person. Even during the pandemic, whenever we used to have a skype meeting, he always made sure that everything with me is fine after the discussion on work so that I can keep my focus on my work.

The second person being significant is my second supervisor, Dr. Gaetano Di Caterina. I am grateful to him as well for becoming my secondary supervisor and putting his trust in me. More than being a supervisor, he always behaved like our friend. Dr. Di Caterina has always helped with reviewing my research writings and provided me valuable feedback to make them better. He also made sure that I participate in various group discussions and presentations related to the work I am doing in my research.

I want to extend my sincere thanks and gratitude to Dr. John J. Soraghan. I remember the first meeting with him when he asked me about my whereabouts, connected with me, and discussed what I would like to do in my Ph.D. He was the person who suggested me to look into Assistive Technologies after Dr. Petropoulakis encouraged and motivated me to take Deep

Learning in Artificial Intelligence as the main focus of my work. Apart from topic selections, he made sure that I get all the necessary resources necessary for my work. Even he helped to me review some of my works. The best part I liked about Dr. Soraghan is that he used to bring different experts from different companies and made me present my work. That helped me to get some valuable feedback and saved much time because that helped me to know what may work and what may not work. Dr. Soraghan has always been very friendly and his smiling face used to fill me with all positivity.

My special thanks to my friend and colleague in CeSIP, Keerati Kaewrak. She had been with me in most of the works. We together have done some of the best works and achieved some remarkable success. Working together has helped us to develop a sound knowledge base that will also help in our future careers. I am also grateful to my colleague Dr. Paul Kirkland who recently defended his Ph.D. thesis successfully, for always helping to set up the required resources appropriately so that the work can be carried out smoothly. Weijie Ke has also been generous support as he was the only other student pursuing his Ph.D. under Dr. Petropoulakis and Dr. Gaetano Di Caterina's guidance. I thank all other colleagues with whom I shared RC351 laboratory over the last couple of years, which directly or indirectly helped me in some way or the other.

I am also grateful to my annual reviewers, Dr. Stephan Weiss and Dr. Paul Murray. They, with the most sincerity, did the review and observed the presentation. Both of them provided essential feedbacks regarding work and presentation. I would like to specially mention Dr. Weiss for providing me his feedback during my first annual review that he attended through video calling from Australia. That proved how much he values anyone’s time and work. He also provided me some material on writing literature reviews in a much better way, which helped me.

I would like to extend my sincere gratitude to the University of Strathclyde’s administration that supported me everywhere, from my accommodation to my laboratory. I have been fortunate enough to study, research and walk through the corridors of the world-class institution that has, over its history of more than 200 years has witnessed some genius luminaries like John Logie Baird, Thomas Graham, James Young, Andrew Ure, etc. working in its abode and walking through the same corridor that led them to the path of some unforgettable success.

I, with folded hands, would like to thank my parents, who sacrificed everything to see their son earn a doctoral degree from one of the world’s finest institutions. I would also like to offer my appreciation to all my school, UG and PG teachers who helped me to gain proper

knowledge. At last, I thank the almighty God for being so kind to me that I met such wonderful people in my journey of life till now. Thanks to everyone once again. Please forgive me if I forgot to mention someone here. If I forgot someone, please remember that you have been equally responsible for my success.

Many thanks and greetings to everyone once again and I dedicate this thesis to everyone responsible for helping me to reach this juncture. My kind regards to all. I pray to God to bless each and everyone.

Abstract

This thesis addresses the fundamental issue of indoor home navigation in Intelligent Assistive Systems (IAS). The issue of inefficient indoor home navigation exists in IAS because of the inefficient indoor home scene and object recognition. The problem is therefore addressed by developing different novel methods using neural networks in this thesis. Apart from addressing the mentioned problem, the developed novel methods also focus on addressing the problems associated with neural networks. The issues related to neural networks addressed in this thesis are the high total number of trainable parameters and the inability of neural networks to produce good accuracy on smaller datasets.

A traditional Capsule Neural Network (CapsNet) is first used to implement indoor homes scene recognition for the first time. The CapsNet produced good accuracy, but it had a very high total number of trainable parameters. This led to the proposed development of NoSquashCapsNet. In NoSquashCapsNet, the squash function is removed from capsules (Capsules are the backbone of CapsNet that helps to acquire orientation of features in vector form), and Max Pool layers are introduced in the architecture. These modifications help to reduce the total number of parameters and remove the restriction of not changing the direction of vectors in capsules. The accuracies produced by both CapsNet and NoSquashCapsNet were lower but comparable with other networks and remained the same in both cases. The accuracies produced were on small datasets. Therefore, from the knowledge gained from implementing CapsNets for indoor home scene recognition, more efficient indoor object recognition networks were developed with more capabilities by restructuring and improving the initial designs

The proposed CapsNets developed for indoor object recognition are 1D CapsNetA and 1D CapsNetB. 1D CapsNets are developed to recognise 3D objects. The use of 3D object datasets makes it easier for CapsNets to capture the orientation of the objects. This will enable an IAS to recognise the objects from any viewpoint. Therefore, this method does not require the conversion of the 3D point cloud dataset to 3D voxel grids. Developing 1D CapsNets and using 3D datasets in 1D array format helps to reduce the total number of trainable parameters and produce comparable accuracy on smaller datasets. Therefore, an efficient system for recognising indoor objects present in any orientation is developed for IAS, enabling an IAS to handle any object when required. However, even if an IAS can now recognize any object present in any orientation, it will never be very useful if it cannot also recognise indoor home scenes properly.

NoSquashCapsNet has produced an accuracy that is adequate for many tasks. However, it is not enough for an IAS expected to perform indoor home assistance for elderly or infirm people. Therefore, Convolutional Neural Networks (CNN) combinations were used to develop efficient indoor home scene recognition. The reason behind using the CNN combination is to perform indoor home scene recognition through multiple object detection. Multiple object detection is performed using transfer learning of a pre-trained Mask-RCNN (Mask-Region based Convolutional Neural Network) because of the instance segmentation performed by Mask-RCNN. Pre-trained Mask-RCNN helps to produce different object combinations for different indoor home scenes. Another CNN is also developed, which could be trained on object combinations produced by the pre-trained Mask-RCNN. The pre-trained Mask-RCNN's output is connected to the newly developed CNN to perform the indoor home scene recognition. The connection produced the Mask-RCNN+CNN combination. Despite being trained on a very small dataset, this uniquely developed CNN combination surpassed all currently available techniques in terms of overall accuracy.

List of Figures

List of Tables

List of Abbreviations

AI – Artificial Intelligence

DL – Deep Learning

MLP - Multi-Layer Perceptron

ANN – Artificial Neural Network

DNN – Deep Neural Network

CNN – Convolutional Neural Network

FC Layer – Fully Connected Layer

Conv Layer – Convolutional Layer

RNN – Recurrent Neural Network

MLVCNN - Multi-loop view CNN

RGB – Red, Green and Blue

ReLU - Rectified Linear Unit

PReLU - Parametric Rectified Linear Unit

SVM – Support Vector Machine

1D – One Dimensional

2D – Two Dimensional

3D – Three Dimensional

FCN – Fully Convolutional Network

ILSVRC - ImageNet Large Scale Visual Recognition Challenge

PCA - Principal Component Analysis

RCNN - Region based Convolutional Neural Network

PReLUNet - Parametric Linear Unit Network

ResNet - Residual Neural Network

NIN - Network In Network

GLM - Generalised Linear Model

CCCP - Cascaded Cross Channel Pooling

BN - Batch Normalization

FPN - Feature Pyramid Network

PSPNet - Pyramid Scene Parsing Network

G-FRNet - Gated Feedback Refinement Network

GAN - Generative Adversarial Network

RoI - Region of Interest

SPPnet - Spatial Pyramid Pooling Network

IoU - Intersection over Union

bbox – Bounding Box

GPU - Graphics Processing Unit

SIFT - Scale-Invariant Features Transform

SURF - Speed Up Robust Features

CLM - CodeBookless Model

MBConv - Mobile Bottleneck Convolution

SON - Self-Organizing Network

SOM - Self-Organizing Map

AE - Auto-Encoder

r-GAN – raw Generative Adversarial Network

l-GAN - latent space Generative Adversarial Network

EMD - Earth's Mover Distance

CD - Chamfer Distance

GMM - Gaussian Mixture Models

CapsNets - Capsule Neural Networks

COOR - Co-occurring frequency of object-to-object relation

SOOR - Sequential representation of object-to-object relation

Contents

Chapter - 1

Introduction

1.1. Preface

This thesis addresses the issue of navigation and object recognition for intelligent assistive systems designed to help elderly and infirm people. Reports on population by the United Nations suggest that by 2050 the population of elderly people across the globe will have expanded substantially (people aged 60 or more and 85 or above are going to be 21.4% and 4.2% respectively) [1, 2]. Similarly, many reports and surveys show that many elderly people prefer to live alone in their own homes rather than use care homes [3]. It is also known that with the increase in age, there is a corresponding increase in various physical ailments that restrict people's mobility. Therefore, taking care of the elderly and infirm citizens becomes very important. In the United Kingdom alone, 2 million people (as of 2021) aged 75 or over live alone at home [4]. Hence, it is clear that alternative solutions must be found for assisting elderly people, especially those with physical ailments, who opt to live alone.

The best alternative solution for assisting elderly people and helping them to avoid unwanted indoor accidents are assistive systems. Assistive systems can help to mitigate many problems faced by elderly people who live alone and have little or no other assistance. From helping to navigate in indoor environments to assisting them in finding different things at home, assistive systems can potentially change the living conditions of elderly and infirm people. However, the level of assistance which can be provided depends on the capability of assistive systems.

As mentioned earlier, current assistive systems have navigational issues owing to inefficient approaches to indoor home scenes [9, 104-115]. In addition, current object recognition procedures are not as efficient as could be. These inefficiencies, mainly (a) inability of systems to recognise objects which may be in various orientations [116-123, 137, 138] and (b) confusing one indoor home scene for another, have been improved in this thesis by developing a number of techniques based on neural network technology.

Several neural networks have been developed to improve the indoor home scene and object recognition in recent years because neural networks have shown good performance in computer vision tasks. Therefore, the prime focus in this thesis is to develop neural networks that are

easy to construct, sufficiently robust, easy to train, able to utilise modern practices in data handling procedures and can produce good accuracy for indoor home scene and object recognition. In this way, the aim is to improve the autonomy of Intelligent Assistive Systems (IAS). As needs and requirements increase, indoor assistive systems need to improve their capabilities in order to provide a reliable and efficient service. Many assistive systems are already available like Alpha-2 [20] and Pillo [21], which can assist people in different ways, such as in receiving medication, other health assistance and entertainment. The available assistive systems are termed Intelligent Assistive Systems because they can act and make decisions on their own using the input they receive. Currently available assistive systems can assist in different ways, but they have limited scope and are not of general-purpose.

The importance of the issues discussed in the above paragraphs can be understood by the existing works, for example, by researchers at the University of Salford [23]. They have created CareBots for indoor assistance, which can carry objects from one room to another, and Mitsubishi's Wakamaru robot [22], which can assist people in receiving their medicine on time. Further, Microsoft has developed an imitation learning process for indoor assistive systems, which learns different activities from people and then reproduces them when required [5]. Microsoft is using AI concepts in developing advanced assistive systems for assisting people with disability [6]. Moreover, the importance of the aforementioned challenges can be easily understood by the initiatives taken by Facebook Artificial Intelligence Research (FAIR). FAIR has shown interest in making robots learn different home indoor tasks like cleaning fridges, cleaning the house, setting up furniture, etc. This is achieved through the introduction of a platform called Habitat 2.0, where robots could be virtually trained for such tasks [7]. Furthermore, in collaboration with Matterport, FAIR has made a large dataset available for academic research [8], which contains 3D indoor home scenes. This clearly shows the intention to expand research in assistive systems using Artificial Intelligence. Therefore, these examples further endorse the need for improving the IAS capability, which is the aim of this thesis.

1.2. Research Motivation

There is an obvious need to increase the navigational capabilities in IAS. Increasing navigation capability will help IAS to increase their scope and autonomy. Increasing scope and autonomy of IAS will further help IAS to offer more and better services to elderly and infirm people. Offering more and better services like bringing a glass of water from kitchen to bedroom, currently is not possible owing to the drawbacks that the available IAS have. The most important drawback is simultaneous recognition of indoor scenes and associated objects.

Both the indoor scene and object recognition tasks are vital for IAS. An indoor IAS is expected to assist a person staying indoors in multiple ways like bringing different objects from one indoor home area to another or letting the person know where an object is kept. Therefore, if the person asks the IAS to acquire an object, the IAS must react to the demand by first recognising the normal environment where the object is expected to be and then recognising and acquiring the object which may not be in its expected orientation.

The problems of IAS in not being able to recognize the scenes is usually due to some of the following reasons:

- It is observed that the accuracy of scene recognition is greater for outdoor scenes than it is for indoor scenes [9]. This can be attributed to the presence of similar objects in different indoor home scenes. For example, a table can be present in many different room scenes within a house. This can give rise to confusion in neural networks when attempting, during the learning process, to differentiate between rooms.
- Having lower accuracy on indoor home recognition is also due to a general lack of large datasets for indoor home scenes. In general, more research emphasis has been given to outdoor scenes and objects. Hence the relatively small size of datasets for indoor objects does not allow for adequate training of neural networks.
- There is also an absence of systems that can produce good accuracy when trained on smaller datasets.
- Currently available systems require multiple views of the same object to recognise the object from any angle. This can lead to very large datasets which are not necessarily easy to process and possible to obtain.
- There is a lack of proper extraction of information on the relationship between different elements in one image. The problem exists because neural networks, when trained for indoor home scene recognition, instead of extracting the information of only important objects present in the scene and the relationship between them, they learn the scene as a whole pattern. For example, if a room scene contains a window, the neural network also learns the pattern of the window. As a window can be present in any room, the neural network may easily get confused between room scenes. This results in false positive outputs.
- Absence of systems with a lesser total number of trainable parameters that can produce better accuracy using smaller datasets. The total number of trainable

parameters must be as low as possible so that the training process of a neural network is faster and easier.

The aforementioned issues are the primary motivation points that initiated this research and the techniques and capabilities presented in this thesis.

1.3. Research Aims

Neural networks have been chosen for accomplishing the indoor home scene and object recognition tasks in this thesis because neural networks have produced better results on computer vision tasks. Neural networks performing the indoor home scene and object recognition can help to expand the IAS navigation capabilities and operation range. Moreover, they will also improve IAS efficiency by making them learn faster and have better accuracy on limited resources (e.g., availability of smaller datasets). Therefore, the following are the aims of this research:

- To implement Capsule Neural Network (CapsNet) for indoor home scene recognition. In indoor home scenes, the arrangement of objects matters as it helps to understand the scene. However, these arrangements could change, but the scene remains the same. CapsNet could help to solve this problem as they are designed to capture object orientation from any viewpoint. This may also help to produce good accuracy on a smaller dataset as the requirement for multiple views may be reduced or eliminated.
- To make neural networks compatible with current dataset structure concepts. Developing such neural networks, or modifying existing ones, might help to learn the orientation of objects and lower the total number of trainable parameters so that the training process gets faster. Orientation extraction may address the issue of data augmentation, which occurs due to taking the images of single objects from different viewpoints (taking images of an object from its every possible viewpoint is also not possible). Further, this may also help in achieving good accuracy on a smaller dataset.
- To develop indoor home scene recognition through object recognition using a combination of neural networks. So that neural networks are enabled to learn only the necessary items or objects that constitute an indoor home scene. Moreover, such a technique could also help to achieve lower trainable parameters with good accuracy on smaller datasets.
- To deploy different suitable Deep Neural Networks (DNN) for training and testing and compare their performance to reach the highest precision level.

1.4. Original Contributions

The work presented in this thesis has led to the following knowledge contribution towards indoor home scene recognition,

i. Implementation of CapsNet for the first time to recognise indoor home scenes (Chapter 4).

ii. The development of a different type of CapsNet structure that contains a capsule without a squash function (Chapter 4) and has Max Pool layer with a convolutional layer for indoor home scene recognition. Motive behind developing such CapsNet architecture is to retain more information and reduce total number of trainable parameters.

iii. Development of two one-dimensional (1D) CapsNets (1-D CapsNetA and 1-D CapsNetB) (Chapter 5). Development of such CapsNet architecture helps in lowering even further the total number of trainable parameters. Even though these are one-dimensional neural network architectures, they are capable of recognizing 3-dimensional (3D) objects. They can also be trained directly on the point cloud form of 3D datasets instead of first converting the 3D datasets to 3D voxel grids. This makes these Neural Networks compatible with current dataset structure concepts.

iv. Development of a combination of neural networks, Mask-RCNN and CNN, to make possible object-assisted indoor home scene recognition using only neural networks. In the developed combinational neural network, a pre-trained Mask-RCNN on COCO dataset is responsible for object detection which produces object combinations. The other part of this combination consists of a CNN which is connected to the Mask-RCNN and which recognises the indoor home scenes from the identified objects. The CNN is one-dimensional. (Chapter 6)

1.5. Thesis Outline

There are 7 chapters in this thesis. The first chapter is the introduction which gives a complete overview of the work presented in this thesis and it explains the aims and knowledge contribution of the presented research. Chapter 2 and 3 contain the review of the related works. Chapter 2 explains why there is a need for assistive systems and provides details on different indoor assistive systems that exist presently. The review helps to understand the different

abilities and deficiencies present in the existing assistive systems. Chapter 3 reviews CapsNet and CNN in detail. The chapter also discusses the different available CapsNets and CNNs available for scene and object recognition. Furthermore, review in the chapter also helps to know why deep learning is the best approach to make assistive systems more efficient and ready for future challenges.

Chapters 4 to 6 present the novelties that help in knowledge contribution. Chapter 4 shows the CapsNet implementation for indoor home scene recognition for the first time. It helps to understand the behaviour of CapsNet on indoor home scene data. The performance of CapsNet is also compared with the performance of Faster RCNN, Fast RCNN and Mask RCNN on the same dataset. The performance of CapsNet on the reduced size of the dataset is also analysed. Further, in Chapter 4, NoSquashCapsNet (Capsule Neural Network with no squash function) is developed which helps in overcoming the discrepancy found in CapsNet for indoor home scene recognition. The NoSquashCapsNet helps to increase the training efficiency by drastically reducing the total number of trainable parameters without decreasing the accuracy. Chapter 5 presents three 1-Dimensional CapsNets which can recognise 3-D objects. The three proposed neural networks are PointCapsNet, 1-D CapsNetA and 1-D CapsNetB. The 3-D dataset is converted into the 1-D array so that the 1-D neural networks can be trained on the point cloud dataset and keep trainable parameters less with improved accuracy.

Chapter 6 presents the implementation of object detection-assisted indoor home scene recognition using a combination of neural networks. The indoor home scene recognition through object detection using neural networks, which till now has remained unexplored, is presented in this chapter. The proposed method in this chapter has shown state-of-the-art performance. The combination of Mask-RCNN and CNN is developed in this chapter, where the Mask-RCNN produces the combination of detected objects in the indoor home scene, and the CNN performs the indoor home scene recognition. The Mask-RCNN and the CNN are interconnected. Chapter-7 presents the conclusions and future scope of the presented work in this thesis.

1.6. List of Publications

i. Amlan Basu, Lykourgos Petropoulakis, Gaetano Di Caterina and John Soraghan, *"Assistive Technology Evolving as Intelligent System."* New Trends in Computational Vision and Bio-inspired Computing, Springer, 2020, pp. 289-303.

ii. Amlan Basu, Lykourgos Petropoulakis, Gaetano Di Caterina and John Soraghan, *"Indoor home scene recognition using capsule neural networks."* Elsevier Procedia Computer Science. January 2020, no. 167, pp. 440-448.

iii. Amlan Basu, Lykourgos Petropoulakis, Gaetano Di Caterina and John Soraghan, *"Modified Capsule Neural Network (Mod-CapsNet) for Indoor Home Scene Recognition."* IEEE International Joint Conference on Neural Networks, July 2020, pp. 1-6.

iv. Amlan Basu, Keerati Kaewrak, Lykourgos Petropoulakis, Gaetano Di Caterina and John Soraghan, *"3-Dimensional object recognition using 1-dimensional capsule neural networks."* IEEE International Conference on Emerging Techniques in Computational Intelligence.

v. Amlan Basu, Keerati Kaewrak, Lykourgos Petropoulakis, Gaetano Di Caterina and John Soraghan, *"Indoor home scene recognition through instance segmentation using a combination of neural networks."* IEEE World Conference on Applied Intelligence and Computing.

Chapter - 2

Indoor Assistive Technology for Elderly and Infirm People

2.1. Introduction

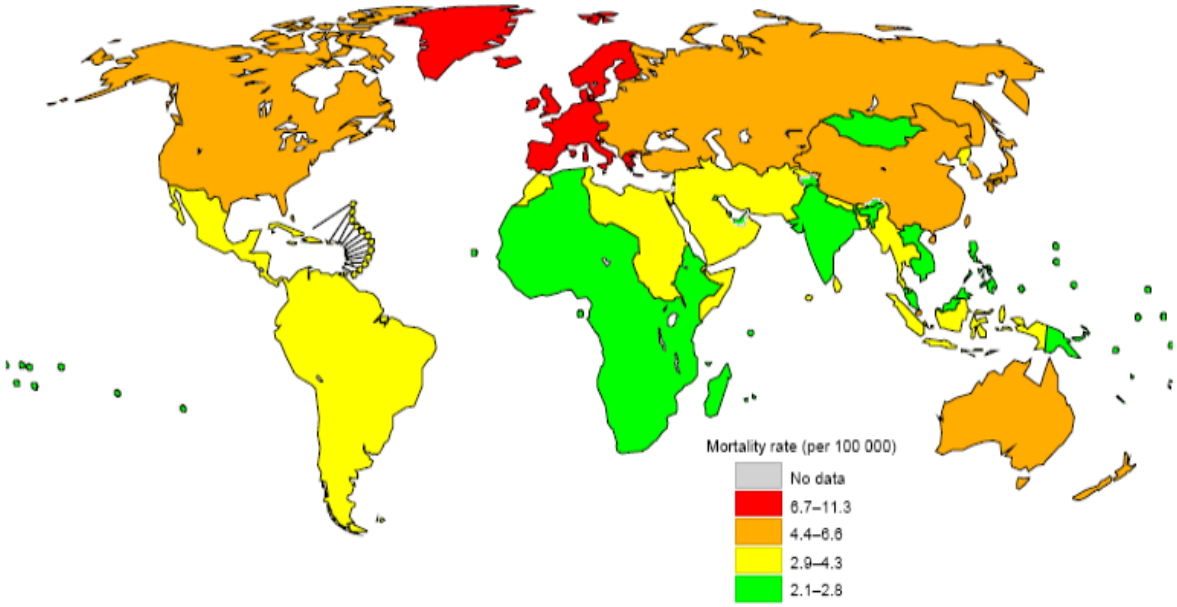
The work that the chapter presents is specifically for indoor assistive systems. In this chapter, various assistive technologies which are associated explicitly with indoor environments are discussed. Current assistive systems, specifically developed for elderly people to help them in indoor environments, are still very basic and purpose specific. They also have too many requirements which include wearable devices that have to be manually operated, non-wearable devices having too much complexity, long installation processes, user input is required for almost every task to be performed, require substantial time to complete tasks and sometimes the systems are not even affordable. To increase the autonomy, range and reliability of such systems, several navigational approaches have been developed comprising different technologies. This will allow assistive robots to extend the scope and purpose of their operations.

This section provides a literature review of the most important of these assistive systems and some of the approaches that have been used to impart intelligence and increase their autonomy. Hence, section 2.2 discusses the challenges that people face currently and will face in future which justifies why there will be a requirement for assistive systems. Section 2.3 discusses available indoor systems, different advanced mobile robots available for indoor assistance are discussed, some fall detection systems currently available and systems used for indoor navigation. Section 2.4 concludes about what future requirements are needed for increasing their intelligence and autonomy.

2.2. Why Assistive Systems are Required?

The world will witness a rapid expansion in the percentage of senior citizens. According to estimates and surveys, it is predicted that by 2050 the people aged 60 or more are going to be 21.4%, and people aged 85 or above will be 4.2% [1, 2]. Both the estimates are double and quadruple of the present percentages, respectively. Physical ailments increase with increase in

age. Even the flexibility to move also decreases in most people because of body composition changes leading to muscle loss and lean body mass, known as sarcopenia. With an increase in age, the bones also become fragile. This makes a person more prone to fractures because people are more vulnerable to physical falls leading to hip fractures. Millions of elderly people having age more than 65 years face falls, as Centres for Disease Control and Prevention reported [10]. It is also reported that one in every three elderly people suffer fall incidents. An elderly person who suffers a fall once is more vulnerable to falling again [11]. Around 2.5 million elderly people worldwide have been treated for fall injuries and the numbers are increasing every day [12]. In Figure 2.1, global mortality due to falls is shown in detail [13].



Fall-related mortality rates (per 100 000 population) in WHO regions, 2000

Africa	Americas		South-East Asia		Europe		Eastern Mediterranean		Western Pacific		
LMIC	HIC	LMIC	India	Other LMIC	HIC	LMIC	HIC	LMIC	HIC	China	Other LMIC
2.7	6.5	3.9	2.1	3.4	11.3	6.6	2.7	4.3	5.3	5.7	2.8

HIC, High-income countries; LMIC, Low- and middle-income countries.

Figure 2.1. Fall related mortality world-wide [13]

Arthritis is another problem that is a prevalent disease among senior citizens. According to WHO (World Health Organisation), 9.6% of men and 18% of women above 60 have a very high tendency to be affected by Osteoarthritis [14]. Osteoarthritis is a disease in which 25% of the people suffering from it cannot perform daily life tasks and 80% of the people suffering from it develop limits in movement [14].

Another prevalent disability among senior citizens is some form of cognitive impairment. In the world, 3% to 19% of the people above 65 have some cognitive impairment [15]. In

cognitive impairment, a person loses the power to think and make decisions for everyday tasks. One of the most rapidly increasing cognitive impairment diseases in the world is Alzheimer's. Worldwide there are 44 million affected by Alzheimer's, as per the report by Alzheimer's News Today [16]. Further, there are approximately 415 million people who have diabetes which is an incurable disease [17]. The complications of diabetes lead to disabilities in a variety of ways, like decline in mobility. Such diseases further increase the risk of fall.

These estimates indicate that there would be a rapid increase in the demand for different indoor assistive systems because an increased number of people choose to live alone. Moreover, it will be difficult to have enough carers for people opting for such service because by 2025 there will be shortage of 1 million carers [18]. Therefore, it becomes important that assistive systems must be appropriately developed with all necessary innovations which can be implemented to solve the problems and drawbacks present in existing IAS (Intelligent Assistive Systems). Moreover, there is a need to make such systems more intelligent and reliable so that the people relying on IAS systems do not require to follow complex instructions to operate them.

2.3. Indoor Systems

Intelligent Assistive Systems (IAS) have a significant role to play in assisting humans in different ways. Indeed, development of IAS have been very slow for many years because of the unavailability of specific firmware and software required for their operations. However, this picture is changing rapidly and it will help the future development of IAS.

An IAS is a system that can use its intelligence and experience to make a particular decision that is expected of it. IAS intelligence and experience are acquired through training using machine learning or deep learning concepts. There are also many systems developed using the concepts of fuzzy logic, fractional calculus, expert system, genetic algorithms, particle swarm optimization and evolutionary computation [141].

There are some IAS present as commercial products for indoor purposes which are also known as Social Assistive Robots (SAR) [19] that help people in academic and commercial exercises, lifestyle improvement, lowering stress, socialization and social sign recognition. These are systems like Pillo, Aido, Alpha-2, Morebot and Miko. For instance, Alpha-2 [20] shown in Figure 2.2, is considered to be one of the most advanced IAS robots. It can mimic different human movements, assists in doing fax and calls, stores voice calls, interact with different people, have general knowledge and it can even be used for purposes like advising on

plumbing solving problems, helps in reminding different tasks, and also assists with the weather forecast and precautionary measures when a person is moving out of the house.

Figure 2.2. Alpha-2 Robot [20]

Another good example to understand IAS is Pillo [21] shown in Figure 2.3. It is an IAS robot that dispenses medicines. It can recognize any person and keep track of medicine timings to be taken by the patient. Also, whenever the medicines go out of stock, it automatically notifies for a refill. It further helps in providing different diet charts, exercise schedules, and advice related to different sicknesses. It quickly can be connected to any smart device. It enables the person to monitor physical activities by providing related information like how many calories the person has burnt and how much more must be burnt. All these works done by an IAS make a person's life a lot easier and, at the same time, make it safer and more prolonged. Pillo is a stationary robot, requiring that the patient needs to be near the robot at the time it dispenses the medication.

Figure 2.3. Pillo Robot for Health Assistance [21]

IAS have many other issues that need proper solutions so their designs can improve in precision and efficiency. At the moment, IAS have the deficiency of the most fundamental

quality of recognizing indoor home scenes and objects. IAS available need to be provided with maps of the indoor area where they are deployed. This increases the time and cost of deployment. Providing maps with coordinates restricts the movement of an IAS because then the IAS can only move to the location for which the coordinates are provided. It is also not practical to provide the coordinates of each and every object of a house because of the dynamic nature of location of objects. Therefore, IAS that can perform scene and object recognition without the use of maps are desirable.

2.3.1. Mobile Robots for Indoor Assistance

Mitsubishi Heavy Industries developed the Wakamaru robot [22] shown in Figure 2.4. It is a mobile robot that can shake hand and interact with humans. Reminding humans about medicines is the crucial task that it performs. However, it cannot dispense medicine like Pillo. For navigation it needs to follow the concerned human.

Figure 2.4. Wakamaru mobile robot developed by Mitsubishi Heavy Industries [22]

There are two CareBots developed by researchers at the University of Salford, U.K. In 2013, CareBot P37 S65 shown in Figure 2.5, was developed by Antonio Espigardeiro. The robot is capable of assisting elderly persons staying indoors. It can remind people about medications and it can even remember the medication by storing a person’s face, can recognise the face, and list all the person's requirements. It can create a link between person and doctor through video calling. The robot can carry a meal to the person as it has a navigation map. People

having dementia problems can be assisted using this robot as it can also be programmed with speech therapy and object recognition [23].

Figure 2.5. CareBot P37 S65 [23]

Dr. Theo Theodoridis developed another robot for indoor assistance for elderly people. The robot, Carrie, introduced in 2018 is shown in Figure 2.6. The robot is capable of recognising and grabbing objects that are shown to it. Carrie is capable of mapping a room and can follow complex commands. It can detect falls, remind people of medicine and even if the gas is left on. The robot is able to navigate using sensors. The robot is fitted with many sensors and actuators to carry out the tasks [24].

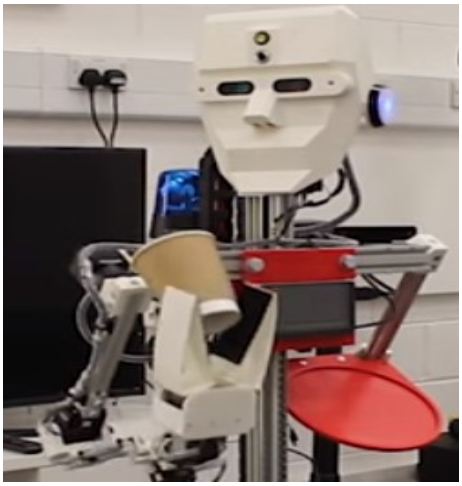


Figure 2.6. Carrie CareBot robot for indoor assistance [24]

BIRON (Bielefeld Robot Companion), shown in Figure 2.7, is another mobile robot for indoor assistance which is developed by modifying PeopleBot of ActiveMedia [25], created in the University of Bielefeld [26]. It can interact with humans and also learns everything from their various inputs, and identify different objects. However, when asked to look for a specific object then it searches for it in the whole house until it finds it [26]. This also implies that BIRON is incapable of performing indoor scene recognition.

COGNRON (Cognitive Robot Companion) [27], shown in Figure 2.8, is a mobile assistive robot that can interact and learn from humans. When required it can fetch the required object to the human. However, it has not been made clear if the fetching can be done from one particular location (say kitchen) to another (say bedroom).

Figure 2.7. BIRON robot for indoor assistance [26]

Figure 2.8. COGNRON robot for indoor assistance [28]

Further, CareBots Carrie [24] and P37 S65 [23], are the most advanced indoor assistive systems which are commercialised despite lacking the basic capability. For example, the Carrie CareBot can grab the objects only when the objects are shown to it in a specific way, but it cannot grab the objects by recognizing them in any orientation on its own. This shows that continuous user input is required. For performing complex works within a room, Carrie has to map the room. So, it takes time to get deployed and start navigating, which indicates that it is incapable of navigating based on indoor scene recognition. P37 S65 CareBot requires more user inputs as compared to Currie. BIRON [26] and COGNRON [28] are also very advanced indoor assistive robots. However, they are still incapable of indoor home scene recognition which increases the time taken to complete tasks. Minimising the time taken to navigate and complete tasks by IAS for assisting elderly and infirm people will help to minimise the movement of people which will ultimately mitigate the cases of falls.

2.3.2. Applications of fall detection and prevention

PERS (Personal Emergency Response System) [29, 30] is an intervention to mitigate and protect elderly people from falls. It provides an emergency button that helps elderly persons to contact emergency services when they experience a fall directly. However, PERS is of no use if a person loses consciousness because of the fall. In fact, Fleming et al. [14] showed, by

conducting a PERS study, that 80% of elderly people who used PERS were unable to use the emergency button when they experienced a fall [31, 32]. The PERS system is shown in Figure 2.9.

Figure 2.9. PERS system working

To overcome the PERS deficiencies, other monitoring systems, which detect falls and automatically contact the emergency services, have been proposed. Such systems have helped to develop a sense of security among users [29]. This also enables users to stay at home longer. Some of the devices that provide such solutions are smartwatches, devices attached to clothing, cameras installed in indoor homes, microphones and pressure sensors on the floor [29, 31].

The fall detection devices can be categorised into two sections, (i) wearable devices and (ii) non-wearable devices. Some of the wearable devices placed on the main body, like the chest, waist and thorax, have accelerometers to detect the changes in speed of movement of elderly people and the motion planes for falls identifications [31].

Elderly people also wear some devices on the head, arms, hands, or feet. Even smartphones have sensing devices like gyroscopes, accelerometers and magnetic field sensors. Therefore, smartphones can also be considered as wearable devices. In smartphones, tri-axial accelerometers are mostly used for fall detection [31].

Non-wearable devices for fall detection are broadly classified into two categories: (i) camera and vision-based devices and (ii) ambient sensor-based devices. Cameras and vision-based devices may include a single camera or multiple cameras. In contrast, ambient sensor-based devices may consist of motion detection sensors like infrared (IR) sensors, pressure sensors, and acoustic sensors to measure various parameters that may help fall detection [31].

Based on data gathered through sensors, artificial intelligence (AI) based algorithms are developed. The falls are detected through body positions like sleeping, sitting and standing. The detection is done using motion analysis, posture analysis, proximity analysis, inactivity,

body shape, and 3D motion analysis of the head [31]. According to Yu et al. [33], to address fall detection, fused with different sensory data is needed so that fall detection can be done accurately.

Fixed or adaptive threshold-based techniques and machine learning (ML) techniques capable of people activity recognition are very much used for fall detection techniques. Threshold techniques can be found in smartphones [34]. Machine learning techniques include support vector machines (SVM), one-class K-nearest neighbour, supervised, semi-supervised and unsupervised ML techniques [34].

In thresholding technique systems, fall detection is done by comparing the sensor data with single or multiple threshold values. Sposaro et al. [35] developed iFall system capable of detecting falls. The system has a smartphone accelerometer and uses adaptive threshold as a detection technique. The system tries to analyse the difference between the data received before and after the suspected fall event. Lopes et al. [36] developed Sensorfall mobile application. This application also uses the accelerometer of the smartphone. Real-time fall detection techniques that perform detection through the measurement of accelerometers are assumed by [36] to be good. Positioning of sensors on the correct part of the body affects the accuracy of fall detection [37]. If the sensor is placed on the hand, which moves quiet frequently, this may lead to false alarms whereas if it is placed on the chest, it will provide alarms when real falls take place. Training data related to falls is required for most fall detection techniques that perform threshold computation using either data analysis techniques or domain knowledge. Some of the future techniques could be developed to prevent falling injuries by calculating the pre-impact of falls and inflating airbags [38, 39]. For fall detection, Zhang et al. [40] use SVM of one-call, trained on non-fall ADL (Activities of Daily Living) outliers.

Further, Hidden Markov Model (HMM) is trained on the static classifiers' probability to improve the person’s activity recognition system. This is based on the ADL, creating a sequence and developing sequential classification algorithms. This technique helps to predict present activity using recent history [41]. Tong et al. [42] trained the HMM on events that occur just before a fall to predict falls for bettering the accuracy. The computation of two thresholds is performed. The data for this purpose is collected using a tri-axial accelerometer which is based on human fall sequences time series.

The tri-axial accelerometer that works on hierarchical HMM methods helps to detect falls and human activities [43]. The same method is improved by placing the accelerometer at the waist or a trouser pocket. The method helped to make the accelerometer independent by

reducing its dependence on the sensor's orientation. Arbitrary sensor placements help to record the signals. Despite this, for sensor orientation, the inputs are made invariant for the classification algorithm [31].

Due to lack of data, some systems predict falls as abnormal activity. Therefore, this is an area of further research as falls are infrequent activities. Yin et al. [44] used one-class SVM trained on different human activities. The abnormal activities are then filtered out in an unsupervised manner. This makes the method a two-stage system. Based on the threshold, abnormal activities are detected by iterating the two-stage method. The method helps to detect abnormal activities and also the false alarms without using labelled data. Falls and slipping simulation in different positions was done to collect the necessary data [44]. The proposed algorithm is shown in Figure 2.10.

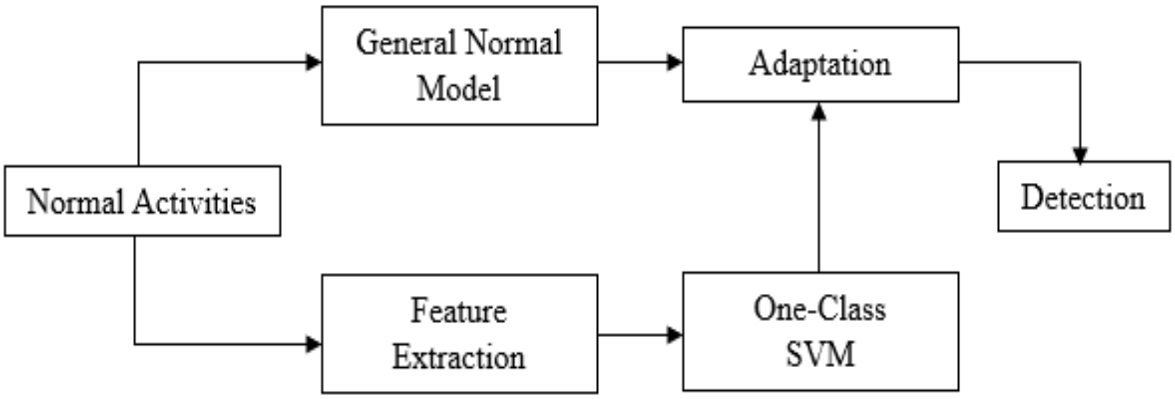


Figure 2.10. Flow chart of algorithm proposed by Yin et al. [44]

Yu et al. [45] used the fall region technique to detect falls using video-based systems. The system identifies if an instance is in the defined fall region or not. To distinguish fall from normal activities like walking, standing, sitting and sleeping. For accomplishing the task, a one-class classification technique is used. Wang et al. [46] developed WiFall to detect falls in the indoor environment. WiFall uses advanced wireless technologies. Wireless technologies help in time variability deployment and CSI's (Channel State Information) special diversity for human activities identification.

Zhang et al. [47] developed Anti-Fall. This is a real-time fall detection system which uses CSI as an indicator for human activities. CSI phase difference of two antennas helps in detecting the falls. Fall-like activities are separated from fall by increasing the accuracy using both phase and amplitude of the system.

Khan et al. [48] try to distinguish falls and non-falls activities using SVM. Normal sound samples are used to extract the required features, which are the data. For fall detection, a

suppressed interference unsupervised acoustic system is developed. The method proves to tackle the unwanted interferences using only two microphones. Parisi et al. [49], through unsupervised learning, try to make a fall detection system by making the system learn about human behaviour. For this purpose, a hierarchical-based self-organising maps (SOM) architecture is developed.

Different assistive systems of different technologies have been presented. In fall detection assistive systems, the most common problem is that the assistive systems react only after a fall has occurred. This shows that these systems do not do anything to protect or minimize the probability of falls. Different fall detection systems also require too many user inputs to react to any situation and keep track of them. Some fall detection techniques even involve multiple cameras, which may be unaffordable.

2.3.3. Indoor Navigation Assistive Systems

A system cannot navigate until it knows where it is and which places are present nearer to it. These two pieces of information cannot be known without the system being able to understand what the environment is like and how it is structured. According to Barber et al. [50], representation of the environment's abstraction influences the navigation system. Geometric, topological and semantic based methods considered as the three main techniques for navigation. In the geometric technique the environment's geometric representation helps sensors and actuators to perform local navigation. In the topological technique the environment's modelling is done using graphs and helps in wider navigation. The semantic technique helps to create information on different elements of an environment using a representative map.

Figure 2.11. Pearl robot for nursing assistance [51]

Further, a robot to determine its current location uses three approaches. These three approaches are, relative position method, absolute position method and environment mapping method. Relative position method is implemented using gyro, accelerometer, or e-compass. The absolute positioning method uses laser, radio waves and beacons to determine the robot's current position. Environment mapping uses a layout of an environment and a camera to determine the location of a robot.

The Pearl robot is a nursebot that helps a person to navigate in hospitals or nursing facilities [51]. Developed by Carnegie Mellon University, it can interact with people by providing advice related to health when required and cognitive support to elderly. Figure 2.11 shows the Pearl robot.

Ando et al. have developed an indoor navigation system called RESIMA [52]. The system specifically helps people with sensory disabilities staying indoors (especially visually impaired). RESIMA assists by using a smart multi-sensor that keeps track of a user's position and deep inertia (psychological state between sleep and wakefulness). RESIMA is a combination of smart paradigms and a network of wireless sensors. The schematic diagram of RESIMA system is shown in Figure 2.12.

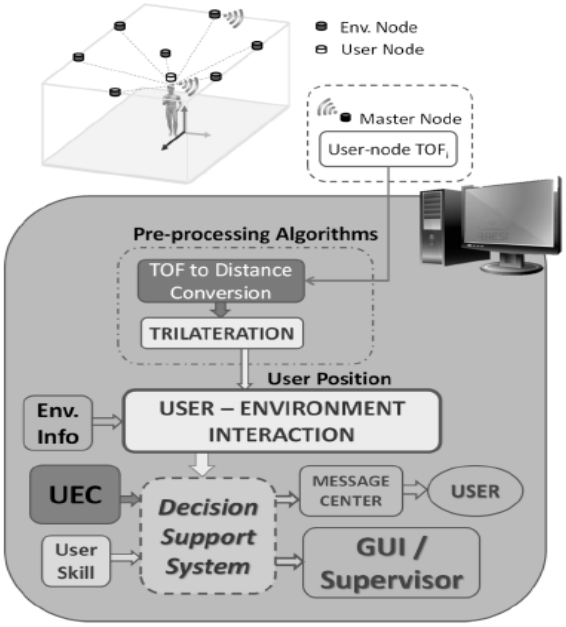


Figure 2.12. Schematic diagram of RESIMA system [52]

Chang et al. [53] developed AssistMote, a wireless sensor network that helps to find ways indoors for any individual. Different components used in AssistMote are the navigation routing engine and user interface called PDAs (Personal Digital Assistance), which help to detect the ambient condition. Dijkstra's shortest path technique is used to develop a navigation algorithm

that also takes ambient intelligence into account. This helps to detect various conditions, including wet floors, while finding the appropriate path to travel. The AssistMote system is shown in Figure 2.13.

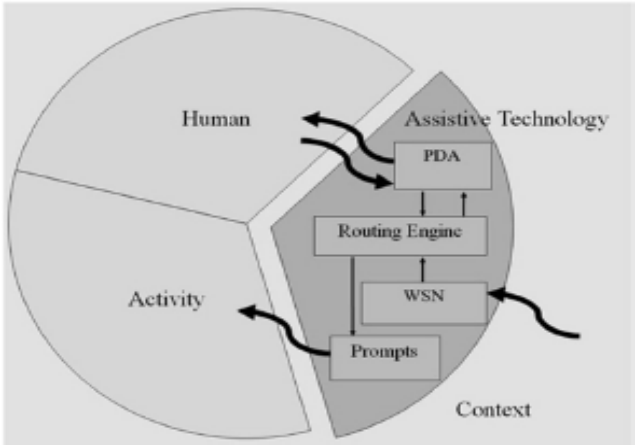


Figure 2.13. Schematic of AssistMote System [53]

Liu et al. [54] developed an indoor wayfinding technology using the Wizard-of-Oz technique. The technique was experimental on many guidance strategies and interface modalities. Accuracy of route completion, completion time and user preference were configured based on various user inputs evaluation. Different routes and modalities like images, texts and audio are included in a counterbalanced design. The working of Wizard-of-Oz techniques is shown through the schematic diagram in Figure 2.14.

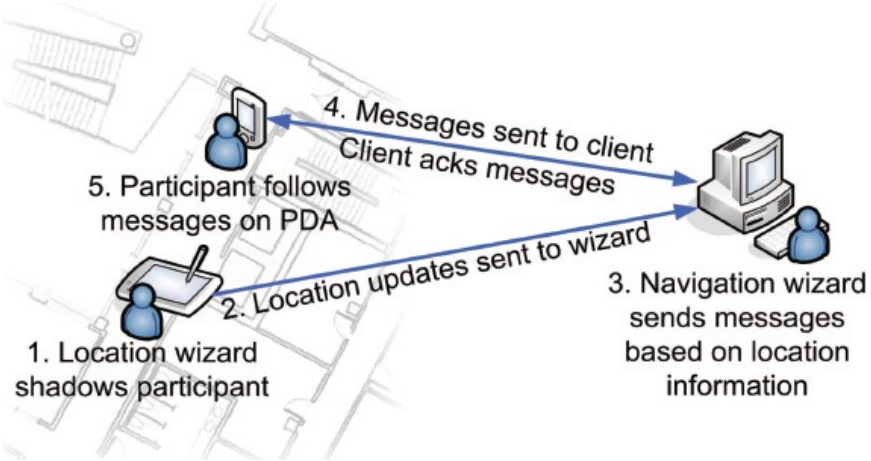


Figure 2.14. Schematic diagram of working of Wizard-of-Oz system [54]

Li et al. [55] developed ISANA (Intelligent Situation Awareness and Navigation Aid) for indoor navigation and obstacle detection for the blind which is shown in Figure 2.15. The indoor map can be edited on the system, whereas different modules are installed in Tango devices. Depth sensors are used to detect different obstacles and is a separate module. ISANA establishes the safest path to the destination for the person. For input and output, a speech-

audio interface is used. The complete system tries to reduce the cognitive load of the concerned person.

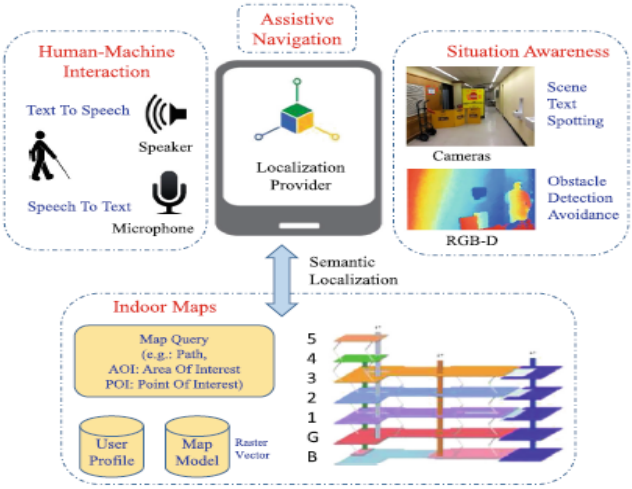


Figure 2.15. Schematic diagram ISANA navigation system [55]

Further, Chang et al. [56] using passive RFID (Radio Frequency Identification) indoor navigation system is developed for people with cognitive impairment. The system developed is for people with Alzheimer's, trauma, cerebral palsy, schizophrenia and mental retardation. The RFID tags have coordinates of different indoor areas and obstacles (x, y, floor). PDAs help to navigate which is connected to all the RFID tags. The RFID tags are placed in different areas and objects indoors. The system is a complete human-computer interface that helps in assisting humans. The complete system architecture is shown in Figure 2.16 which also shows how the system works.

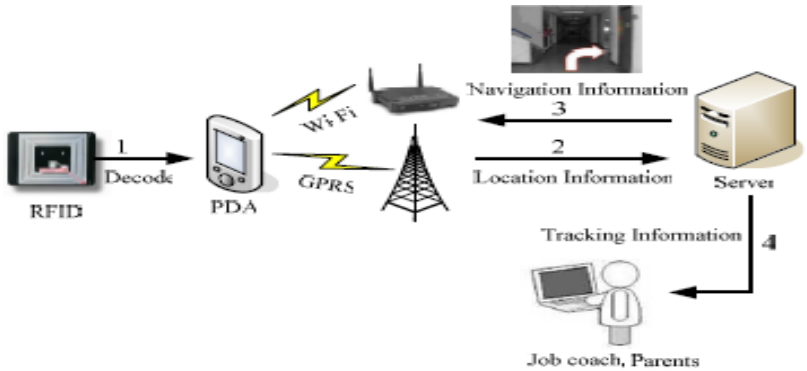


Figure 2.16. Architecture for indoor navigation using RFID and PDA [56]

Holly A. Yanco introduced Wheelesly that is an automatic wheelchair system [57]. The wheelchair consists of a joystick that helps the user to navigate and has a customizable onboard computer and GUI (Graphic User Interface). The device is entirely dependent on the input provided by the user to operate and navigate. Figure 2.17 shows Wheelesly used by a user.

Figure 2.17. Wheelesly used by a user [57]

Caffo et al. [58] developed orientation-based strategies for indoor navigation for people with Alzheimer's disease. The two strategies are Assistive Technology (AT) program and the Backward Chaining (BC) procedure. AT program proved to be more efficient in accomplishing the required task. The AT program involves the use of sound or light devices that are remotely controlled, whereas the BC procedure involves verbal instructions. In the AT program, a source with sound and light is placed at different places controlled by a person using a transmitter. The AT operator provides input to the person needing navigation instructions. In this procedure, the person is expected to move to the location from where the sound and light are emitted. This makes indoor navigation possible.

Kahraman et al. [59] developed an intelligent system for visually impaired people for indoor navigation and guidance. For complex indoor environments, a software prototype is developed with RFID or BLE (Bluetooth Low Energy) infrastructure. In this system, the user provides a specific location input using a special interface which becomes the destination for the system. The system based on destination provides navigation instructions to the user using path optimization procedures like the traveling salesman problem and instantaneous real-time instructions. Figure 2.18 shows the different components installed for indoor navigation of visually impaired.

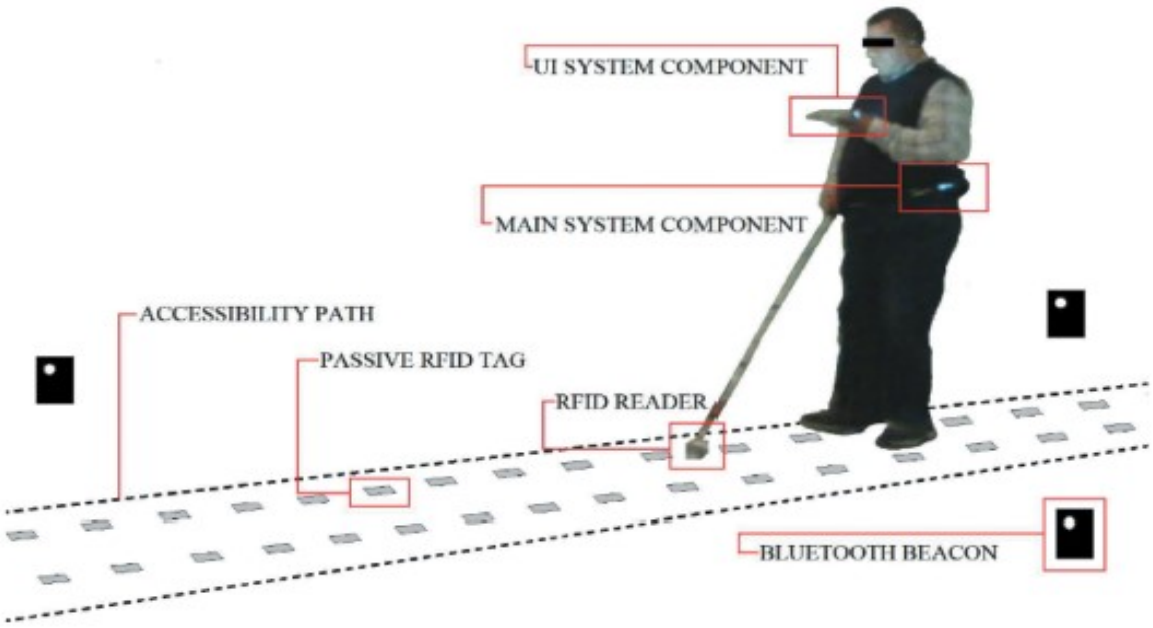


Figure 2.18. RFID or BLE components for visually impaired [59]

Ballestin et al. [60] developed an indoor navigation robot that uses different sensors like laser, scanner and camera. to perform environment mapping. The robot works on three principles to navigate; (i) creating maps of the environment using a fusion of sensors, (ii) obstacle avoidance to reach the goal location and (iii) avoiding obstacles by performing obstacle detection during navigation. The technique tries to map an environment in three-dimension using the Seeker Jr. robot by Adept MobileRobots, shown in Figure 2.19.

Figure 2.19. Seeker Jr. Robot by Adept MobileRobots [60]

Yen et al. [61] implemented a technique in a humanoid that tries to learn spatial knowledge using a mounted cameras. The mounted camera captures the navigation pattern of a human within a room and the humanoid learns the navigation from captured information. The system

can also detect objects. However, as demonstrated, the system is confined to small area of operation.

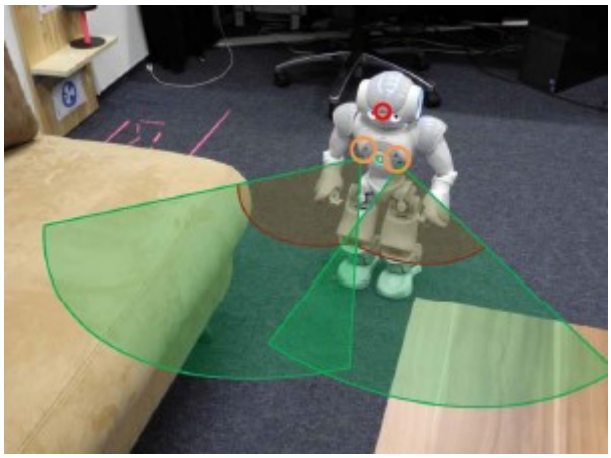


Figure 2.20. Humanoid used by Wenjie et al. [61]

Surmann et al. [62] used deep reinforcement learning in a robot for indoor home navigation. The deep reinforcement learning helps the robot to learn on its own about navigation in an unknown indoor environment. The robot does not have any map or planner. A two-dimensional laser and an RGB-D (Red Green Blue-Depth) camera are used to get the required inputs. Linear and angular velocities of the robot is the Asynchronous Advantage Actor-Critic (GA3C) output. Pre-trained navigator/controller is used to speed up the process and to avoid overfitting problems; random Gaussian noise is used in the inputs acquired from two-dimensional laser to train small networks. The robot needs to navigate in several different environments to acquire the highest level of precision.

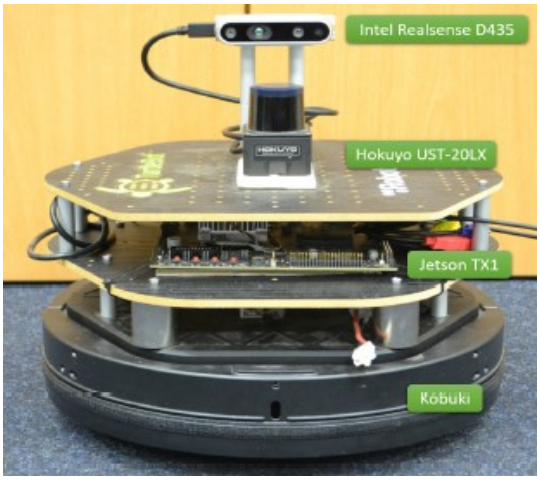


Figure 2.21. Robot implemented using deep reinforcement learning [62]

The indoor navigation systems discussed appear to have issues with creating indoor maps. Such systems need to be provided with new maps every time, which increases the deployment and installation time. Some systems (especially for Alzheimer's assistance) require a person to

continuously operate the system, which, for an Alzheimer's patient, is not an easy task. For many of these assistive systems, people are expected to wear a device and sometimes, use their smartphones to access or control the assisting devices. This becomes very impractical because while indoors, a person cannot be expected to always wear a device or keep the smartphone always with them. Even machine learning techniques, such as SVM, have only been able to detect falls but not stopping them from happening.

None of the systems discussed have been found to be generally able to mitigate the accidents that elderly people staying indoors may face. Most of the systems installed react only after accidents have taken place. Even the systems that try to provide safe navigation to elderly people indoors need continuous user input. This makes the handling of these systems somewhat complex and also a time-consuming process. Such systems can even become a reason for user frustration, which again is a problem that may negatively affect the user's health.

Therefore, devices which can assist the elderly indoors with a minimum number of commands (preferably voice commands) and complete the commands with no further user input are required. Such technology is not in existence because there are no systems displaying good performance in indoor home scene and object recognition. Without indoor home scene and object recognition, a complete indoor assistance system cannot function adequately and efficiently. If an assistive device can perform scene recognition, then indoor navigation becomes easier. Such systems will never require unique coordinate maps of indoor areas to navigate. Therefore, this will make assistive systems capable of getting deployed anywhere in the minimum of time. Such navigation capability in assistive systems can further help to reduce fall rates by cutting down the unnecessary movement of elderly people indoors. Object detection is also required to easily recognise any object that can be present in any orientation. Without proper object detection, an assistive system cannot be expected to assist elderly people indoors efficiently.

As pointed out in [63, 64], the best way to develop scene and object recognition is by using Deep Neural Network (DNN). The reason behind using DNN for this task is because of their proven efficiency on image, scene and object recognition tasks. DNN also help to develop higher intelligent systems and work on minimum user commands. Many different types of DNN have been developed over time specifically for scene and object recognition. Currently the performance accuracy of DNN is high for external scenes but very low for indoor scenes, which is the most common problem found in DNN in this particular context. This is because of the similarity between different indoor home scenes. The similarity is created because of the

presence of similar objects in, for example, different locations of a house. This can be understood from an example of a chair which can be found in the dining room, living room and bedroom of a house. This phenomenon makes it difficult for any DNN system to learn the different locations of indoor homes. There is also a lack of availability of large indoor home scenes datasets. Hence, training of DNN on smaller datasets result in low accuracy.

2.4. Conclusion

In this chapter, different indoor assistive systems like mobile robots, indoor navigation system and different fall detection systems were reviewed. The improvements required in techniques/technologies associated with systems were also discussed in this chapter. In the next chapter, different deep neural networks are explored to find solutions to the navigation issues discussed in this chapter. Deep Neural Networks will be explored as they are supposed to have performed the best among all available techniques in case of scene and object recognition tasks.

Chapter - 3

Application of Deep Learning for Scene and Object Recognition

3.1. Introduction

In this chapter, Machine Learning (ML) approaches focusing on deep neural networks, are examined for scene and object recognition. This chapter concentrates on some general outline concepts relating to some Deep Neural Networks (DNN) and in particular Convolutional Neural Networks (CNN) and CapsNets. The literature review of DNNs is applied to image recognition is also evaluated. Hence, section 3.2 discusses Deep Learning (DL). Section 3.3 is about architecture of CNN and different functions involved in it, different CNN developed and implemented for image recognition, Fast and Faster RCNN and Mask-RCNN are discussed, different scene recognition tasks are discussed that involves both CNN and non-CNN way of implementation and different 3D object detection works using different CNN. Section 3.4 is about Capsule Neural Network (CapsNet), the basic architecture of CapsNet and different CapsNet architectures used in 3D object recognition. Section 3.5 summarises the datasets and Section 3.6 is the conclusions.

3.2. Deep Learning (DL)

Deep Learning (DL) is a sub-category of Machine Learning (ML). Before the introduction of DL machine learning required pre-processing of the data so as to extract features which then can be used to teach a system to classify the raw data as for example to learn about the objects present in an image [65, 66]. DL eased this process considerably as it incorporates representation learning as part of its structure. Deep learning also helps to represent very complex functions. Both these properties, can happen because of the presence of multiple levels of representation, developed using simple yet non-linear modules, which help to transform the representation at a lower level to a representation at a higher level.

The higher-level representation is responsible for reflecting the exact form of input by using all the information acquired from the lower levels. The higher level helps to magnify the important properties and discard the insignificant properties to make the classification task

easier. For instance, in an image, which is an array of pixel values, the first layer tries to learn about the edges present in different parts of the image. The second layer learns about the patterns of edges by identifying them. The third layer combines all the patterns to form a similar input image. Then the subsequent layers are responsible for the detection of objects using the already learned patterns. The interesting point in deep learning is that a developer does not need to design the feature extracting layers. Instead, these are learned using a procedure of general-purpose learning [65, 66].

DL has helped to solve some of major problems which were not possible previously. It has helped to find out complex features present in high-dimensional data. Therefore, it has shown remarkably good performance compared to other ML methods in areas such as image and speech recognition, prediction of the effects of mutation in non-coded DNA on disease and gene expression, prediction of drug molecule activities, natural language processing, sentimental analysis and language translation [65, 66].

3.3. Convolutional Neural Network (CNN)

CNN can be defined as a kind of neural network responsible for processing data with grid topology, like an image. The different applications of CNN are image and video recognition, image classification, image segmentation, natural language processing, financial time series and medical image analysis. CNNs are widely used because the CNNs treat data as spatial information [67]. It is also known for its ability to reduce the trainable parameters because of the presence of three essential features:

1. Sparse Interaction: In CNN, sparse interaction is established by having smaller kernel size than the input image. This helps to extract meaningful information from tens and thousands of pixels of an image that has millions of pixels. This helps to improve the statistical efficiency of CNN.

2. Parameter Sharing: It is a way in which using a particular weighted feature maps are shared. It is done using local connectivity, which helps to connect kernels/filters to all inputs but in sliding manner. Parameter sharing is also called weight sharing.

3. Equivariant Representation: Let us say there is an image x. Convolutional operation is represented as f, and translation operation is represented as g. Now, it is known that convolution is equivariant to translation. Therefore, the result obtained by first convolving x and then translating it is similar to the result obtained by first translating x and then convolving it. Mathematically it can be represented as, $f\big(g(x)\big) = g(f(x))$. Across different data, the

equivariance in convolution with respect to translation in sharing parameters is used. For example, during the training CNN using an image, the first convolutional layer extracts the features. However, the image may have the same feature in different areas on it. One parameter is used to represent extracted features that are similar, which makes the representation of extracted similar features simple for CNN [67].

3.3.1. CNN Architecture

The CNN layers can be broadly classified into four parts: convolutional layers, activation layers, pooling layers and Fully Connected (FC) layers. The vital part of CNN is the convolutional layer, as it uses filters to convolve the data. As the word suggests, convolution is responsible for the filtering process and is always the first CNN layer. It helps in extracting the features from the input. In CNN architecture, the layers closer to the input layer are responsible for simple feature extraction from the images like the edges, curves and lines. In contrast, the layers closer to the output layer are responsible for gathering the information generated by a deeper layer and combining them to extract the high-level features [67].

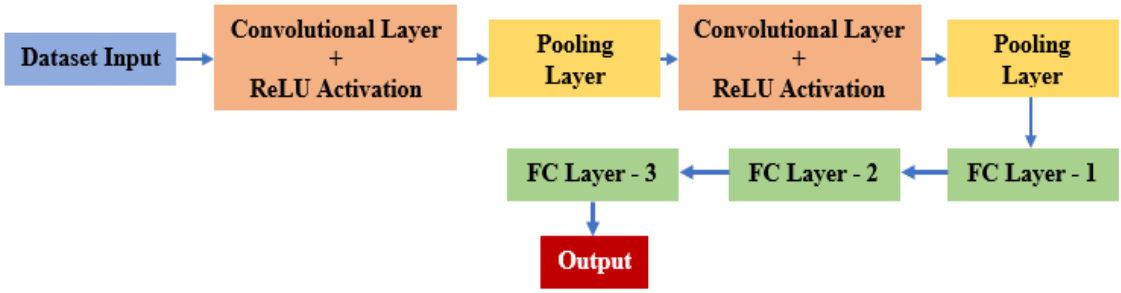


Figure 3.1. Block diagram is portraying the basic CNN architecture to understand the CNN properly. This architecture contains two convolutional layers, two max-pooling layers, and fully connected layers.

Figure 3.1 shows the block diagram of basic CNN architecture with a convolutional layer with the ReLU (Rectified Linear Unit) activation function [68], the pooling layer, followed by another convolutional layer with the ReLU activation function and pooling layer and the FC (Fully Connected) layers.

The convolutional layer is considered the core block of any CNN. It has some filters applied to the given input image and then creates different activation features in that input image. The parameter of the input is w(width) x h(height) x depth(d). For grayscale image the depth is 1 and for RGB image the depth is 3. The convolutional layer has kernels/ filters of any dimension with the same depth as that of input. Across the data, filters are applied using sliding window. Filter depth is the same as input, i.e., RGB format data with filter depth 3 (for RGB image, for grayscale image it will be 1) is applied to data. The images and filters' product performs

element-wise, whose values are summed up for performing the sliding operation in convolution. 2D matrix is produced as an output of convolution with 3D filter processing RGB image. From image data, there is an input tensor of multi-dimension, which requires kernel tensor of multi-dimension. To understand this, consider an image input I and kernel h. The convolution of I and h is written as [69],

$$(I * h)(x, y) = \int_0^x \int_0^y I(i, j).h(x - i, y - j) \quad (1)$$

$$(h * I)(x, y) = \int_0^x \int_0^y I(x - i, y - j).h(i, j) \quad (2)$$

The convolution equation shown in equation (1) helps to perform the convolution by sliding the image over the kernel. In contrast, equation (2) performs the convolution by sliding the kernel over the image. There are more values of x and y in the image as compared to the kernel. Therefore, the convolution equation (2) is used. The resultant is a scalar output. The convolution process is repeated for each value of x and y present in the image [69].

Then, the convolutional layer has activation function. Convolutional layers process the input and give an output with spatial dimension. The output size from convolutional layer is calculated using the following mathematical formula,

$$O = \frac{(W-K+2P)}{S} + 1 \quad (3)$$

In equation (3), O is height/length output, W is the height/length input, K is the filter, P is padding and S is stride. The way of processing the pixels depends on strides. Stride value controls the way filters slide on input for scanning pixels. If stride is 1 then filter slides 1 pixel at a time and if stride is 2 then filter slides 2 pixels at a time and so on. Activation function used is non-linear. This is because the linear functions are not capable of helping neural network in the learning process. To understand this, let us say that A1 and A2 are two subsequent convolutional layers applied on X in the absence of a non-linear activation function. This can be understood mathematically by equation (4) [69],

$$A_1 * (A_2 * X) = (A_1 * A_2) * X = A * X \quad (4)$$

Convolution's associative property makes two subsequent layers on convolution act as single. This also is the same in the case of ANN (Artificial Neural Network). ANN with 100 hidden layers, but without activation, the function is equivalent to an ANN with a single layer.

3.3.2. Different CNN (Convolutional Neural Networks) for image recognition

Once the DNN were unable to process larger dataset and also became slow then the researchers proposed several CNN architectures for different tasks. Image recognition task using CNN is discussed in this subsection so that the CNN's behaviour on capturing the information from images is understood.

The first CNN was proposed by Yann LeCun [70]. The architecture proposed is known as LeNet-5 and is shown in Figure 3.2. The development of such architecture was for recognizing the handwritten digits and converting them into typed computer numbers. It was specifically for the cheques deposited in banks. The architecture had 60,000 parameters. The architecture of LeNet-5 consisted of 7 layers.

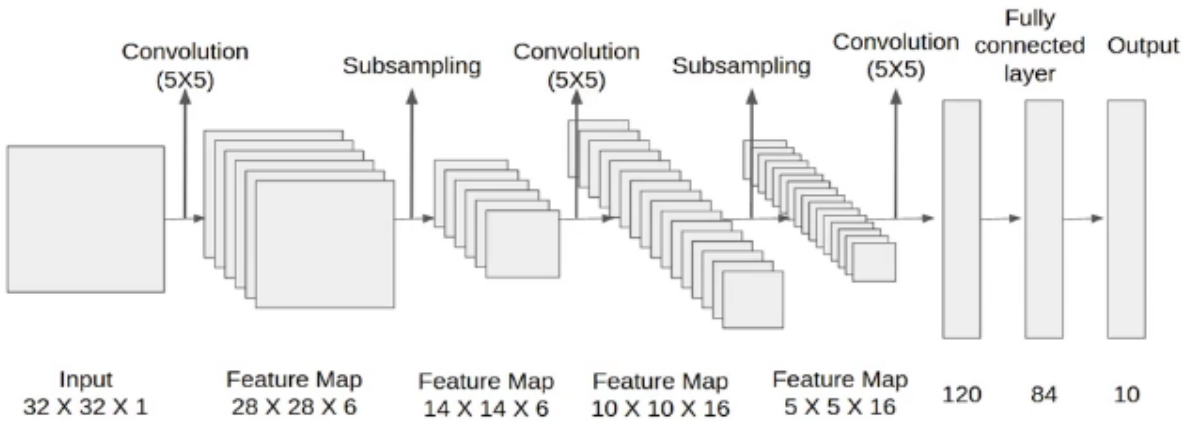


Figure 3.2. Architecture of LeNet-5 CNN [70]

However, after the invention of LeNet-5, the development became slow in this particular field until AlexNet appeared [71]. It is a CNN architecture with eight layers. In ILSVRC (ImageNet Large Scale Visual Recognition Challenge) 2012, it showed exemplary performance and secured the first position with top five error rate of 15.3%. The accuracy shown by AlexNet was astonishing for the people associated with computer vision, and this became the turning point from where CNN became the most commonly used architecture for different deep learning purposes. In Alexnet, to solve the problem of overfitting, the dropout techniques and data augmentation are used. The augmentation is performed to adjust two things; (a) the image intensity using PCA (Principal Component Analysis) and (b) to extract the 224x224 patch from the original image of 256x256 and also from its upside-down orientation (This process is also known as cropping). The five patches are extracted from the four corners and centre of the image. Further, the softmax average is deduced.

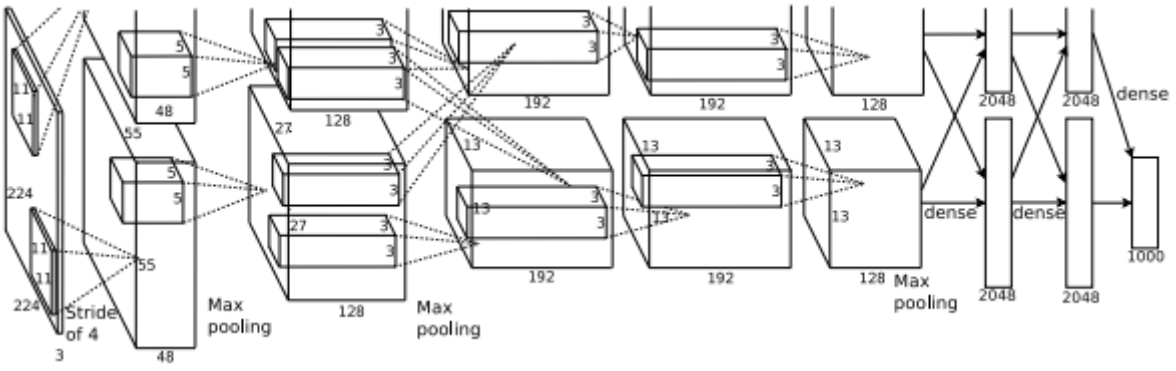


Figure 3.3. Architecture of AlexNet CNN [71]

The same accuracy of AlexNet was achieved by SqueezeNet [72] CNN architecture, shown in Figure 3.4, which claimed to be three times faster than AlexNet and also claimed to have 50 times lesser parameters and model size less than 0.5MB as compared to AlexNet. The squeezenet architecture uses 1x1 pointwise filters instead of 3x3 filters. Using 1x1 filters reduce the depth and computation. Also, to attain a large feature map, there is a huge delay in down-sample. The backbone of SqueezeNet is the fire module that has two layers in it, squeeze layer and expand layer.

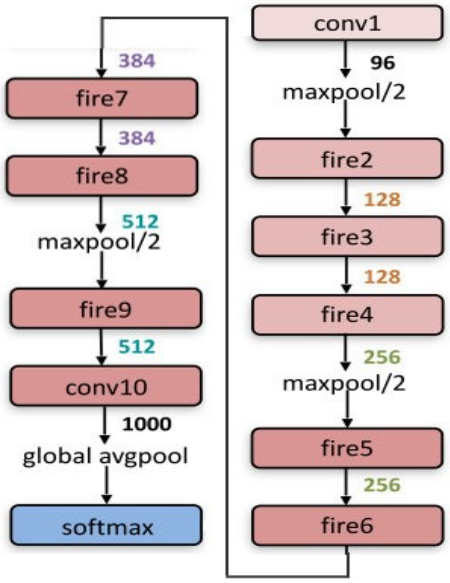


Figure 3.4. Basic architecture of SqueezeNet CNN [72]

In SqueezeNet architecture, there is the stacking of a bunch of fire modules along with pooling layers. The fire module helps in keeping the feature map size intact. The feature map size remains intact because the squeeze layer in fire module first reduces the depth and the expand layer again increases it. The squeezing and expansion mechanism in neural architectures are common. Also, the depth increases with reduction in feature map size to achieve abstract of a higher level. The squeeze module contains only 1x1 filters which indicate that it works like

the FC layers that focuses on same position feature points. SqueezeNet reduces the feature map's depth which enables the expansion layer with 3x3 size filter to less computation, and this increases the speed of 3x3 filter by nine times as compared to the 1x1 filter. However, if squeezing is done on a large scale, then that reduces the amount of information flow and also use of fewer numbers of 3x3 filters affect the spatial resolution by reducing it [72].

A modified AlexNet architecture called ZF Net appeared [73], shown in Figure 3.5. It participated in ILSVRC 2013. The proposed architecture achieved top 5 error rate of 11.2% and placed itself in the first position. The ZF Net architecture is very similar to the architecture of AlexNet. However, ZF Net still has some modifications like the filters used in AlexNet's first layer, the size of the filter is 11x11 whereas in ZF Net it is 7x7. The motivation behind the reduction of filter size was to loose the least amount of pixel information as the 11x11 sized filter lost many relevant pixels information. The first convolutional layer is considered the most important in acquiring the first information on any data [73].

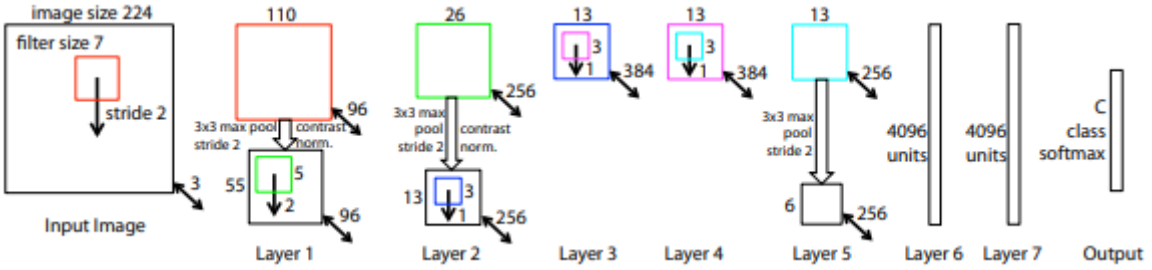


Figure 3.5. Architecture of ZF Net CNN [73]

Moreover, AlexNet showed lesser accuracy even after getting trained with a dataset of 15 million images. Still, ZF Net showed better performance than AlexNet even after getting trained on a dataset of 1.3 million images which is lesser than the dataset used for AlexNet. This also helped to develop the technique of deconvnet (Deconvolutional Network), which helps to map the feature information with the pixel values and is the opposite mechanism to that of CNN. ZF Net helped to provide the real information behind the mechanism of visualization using CNN. The information helped in understanding the architecture and also the further development that is possible in CNN architectures [73].

Further to improve CNN accuracy in image recognition, three CNN architectures proposed are VGG-16, VGG-19, and GoogLeNet. VGG-16 and VGG-19 are the CNNs with 16 and 19 layers, respectively. The architecture of VGG-16 and VGG-19 are shown in Figure 3.6 and Figure 3.7, respectively. The VGG stands for Visual Geometry Group of the University of Oxford. The VGG was able to achieve an error rate of 7.3% [74].

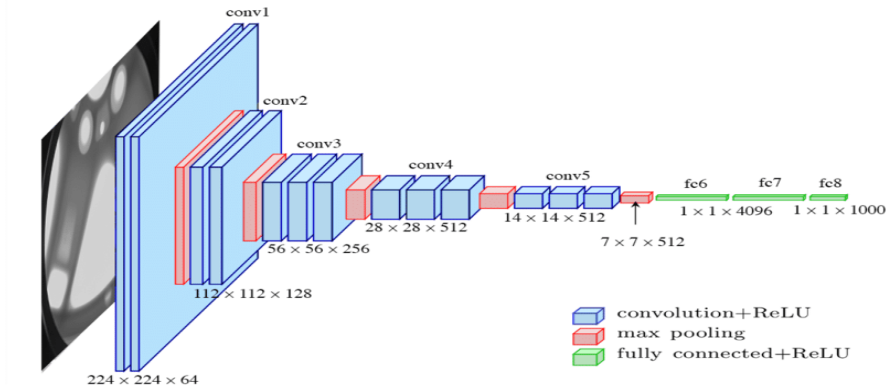


Figure 3.6. Architecture of VGG-16 CNN [74]

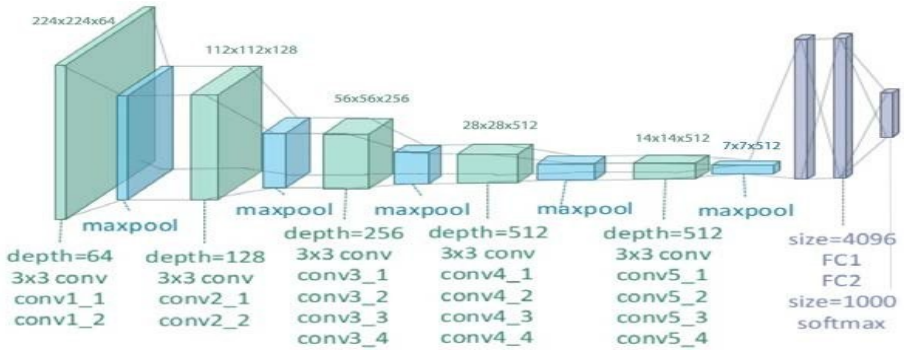


Figure 3.7. Architecture of VGG-19 CNN [74]

However, it achieved the second position in ILSVRC 2014. It has a very simple architecture. In the proposed CNN, there are 3x3 filters with 2x2 max-pooling layers that have stride 2. The use of 3x3 filters in the first layer is for achieving a 5x5 receptive field, which is very effective. Also, the number of parameters decrease, and using two convolutional layers helps to utilize two ReLU activation functions instead of only one. Using three convolutional layers, one after the other, also created a receptive field of 7x7. Spatial dimension shrinks in this network architecture with growing depth because, after every max-pooling layer, the filters double in number. Since there is an increase in the filter size, the volume depth of input increases and the input volume's spatial size decreases. The network performed very accurately in both image classification and localization. While training, the VGG network scale jittering technique was used for data augmentation. The data augmentation done in VGG is the same as it is in AlexNet. However, it also performs image-scaling [74].

GoogLeNet is the CNN architecture with 22 layers [75], shown in Figure 3.8. This 22-layer architecture consists of 9 inception modules (Inception module enables to parallelly perform multiple convolution and pooling operations with different filter sizes). The FC layers are absent in this architecture and to save the parameters, average pooling, linear and softmax are used. For the detection purpose, RCNN (Region based Convolutional Neural Network) concept

is implemented [76], whose architecture in shown in Figure 3.9. In ILSVRC 2014, it achieved the first rank and had a top 5 error rate of 6.67%.

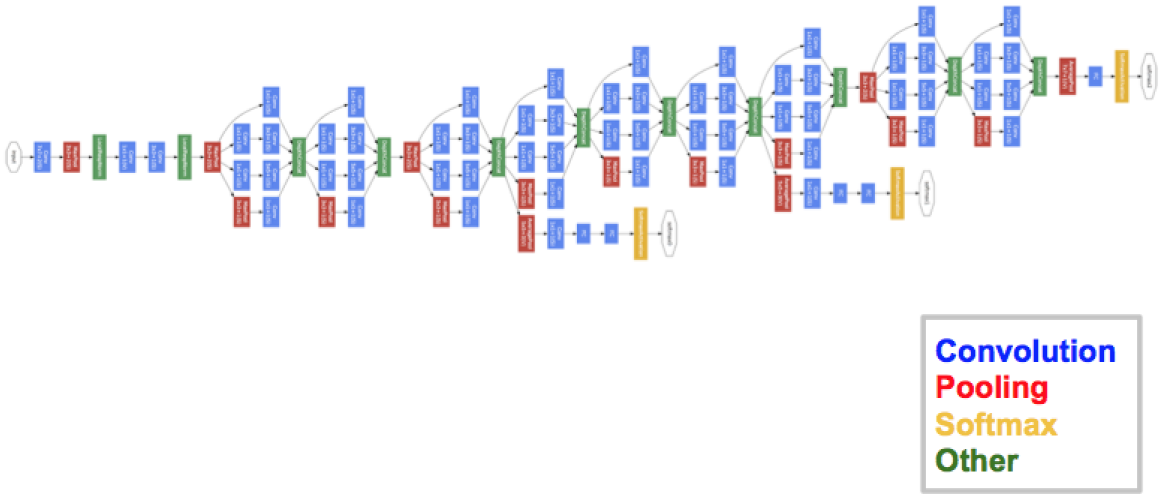


Figure 3.8. Architecture of GoogLeNet CNN [75]

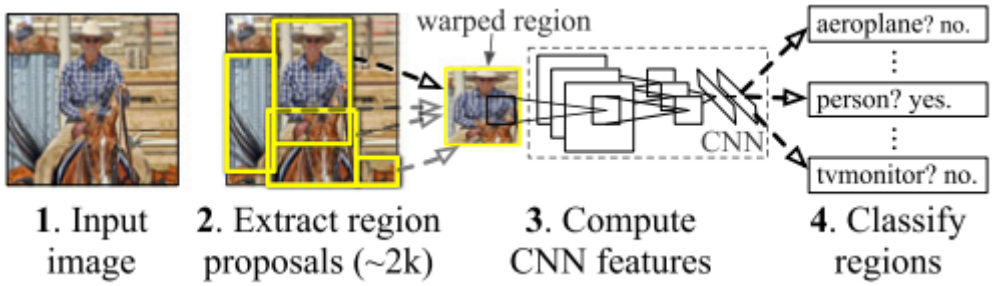


Figure 3.9. Architecture of RCNN [76]

RCNN architecture is partially parallel. The parallel processing in the network is because of the implementation of the inception module. The advantage of having an inception module in a network is that unlike the sequential network, whether to have a pooling layer or convolutional layer is not important because these operations occur in parallel in such networks. However, this kind of implementation was not efficient enough because the parallel process produces many outputs, which leads to a larger channel depth for output size. The first layer uses the filter of size 1x1, which reduces the input size. Thus, the subsequent layers of size 3x3 and 5x5 do not have to deal with larger size inputs. This architecture can extract even minute details of the data and covers a larger area of the data. Furthermore, it solves the problem of overfitting and spatial sizes using the pooling layers easily. Moreover, even after so many operations, the network does not face any problem in its computational performance [75].

PReLUNet (Parametric Linear Unit Network) became the first CNN to surpass the human error rate of 5.1%. It registered top-5 error rate of 4.94% [77]. PReLU used in the CNN is to have a robust initialization method to improve the learning in a deep rectifier network. As initialization is one of the main factors of implementing proper learning, while using PReLU in the neural network, it can be observed that the first convolutional layer has a coefficient which is larger than zero as PReLU helped in acquiring both positive and negative values.

Moreover, the value of coefficient descends with the increase in layers in channel-wise observation. This means that the focus on discernment increase with an increase in the layers, rather than focusing on larger information as it is in the first layer. Weight initialization earlier was done using the Gaussian Distribution. If a fixed standard deviation value is used in a deep neural network with layers of more than eight, then the convergence might not happen properly. The VGG net tried to solve the initialization problem by first making a small network learn and then transfer its learned weights to the larger network. However, this strategy also gave rise to local optima's problem because of increased learning time. GoogLeNet tried weight initialization by constructing a network that converges the auxiliary classifiers in addition to the softmax layer. Further, the Xavier initialization method was proposed, which according to the number of neurons, scales the value [78]. However, since it was designed only for linear activation, it is not suitable for networks using ReLU activation to produce non-linear functions. Therefore, in [77], Kaiming initialization was introduced such that it works in networks with ReLU activation and reports good convergence in networks with more than 30 layers.

Another CNN that came into existence is ResNet (Residual Neural Network) [79]. This architecture has advanced modules of inception. The ResNet contains many batch-normalization and skip connections (grated units or grated recurrent units). Also, residual blocks help in the gradient's smooth flow during the backward pass of backpropagation as other operations are executing at the same time and the gradient is distributed. The interesting fact about the architecture is that the reduction of an image's spatial size takes place only after the first two layers, i.e., from 224x224 to 56x56. In ResNet, there is a residual block in which input goes through a convolution-ReLU-convolution series and the result from convolution-ReLU-convolution series is added to the input. Therefore, unlike traditional CNNs, it also keeps information of the original input and the output that it received. ResNet in 2015 was introduced using the concept of 4 different sizes (in terms of the number of layers), 34 (shown in Figure 3.10), 50, 101, and 152.

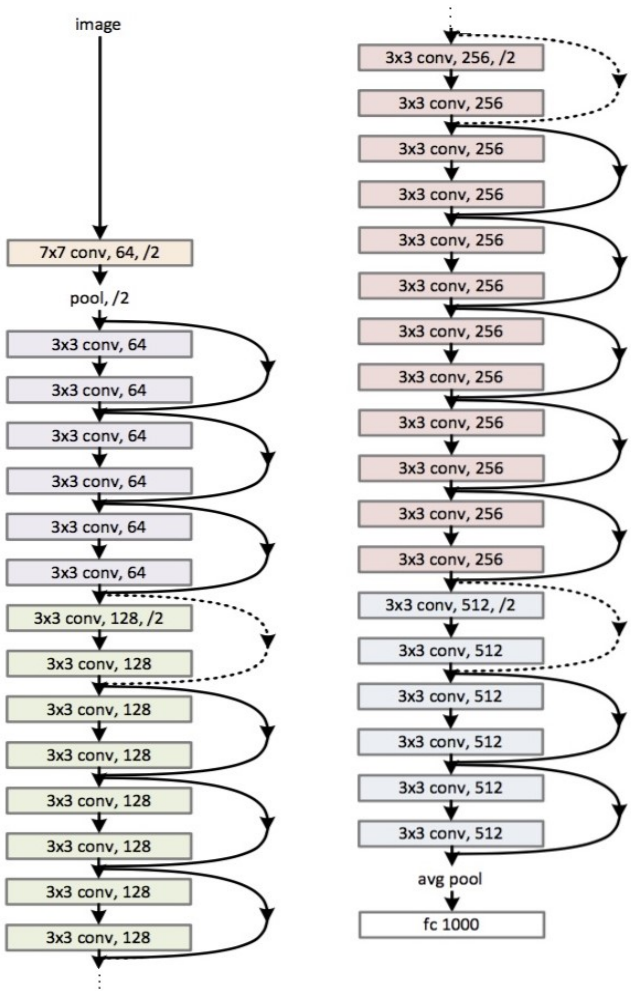


Figure 3.10. Architecture of ResNet 34 layers [79]

The layers in ResNet were further increased to 200 and 1001 [82]. The inception v3 and v4 were modified so that they can be used in ResNet architecture [80, 81]. During the development of deeper CNNs, it is noticed that after some point of deepening of the network, the accuracy levels get saturated. Thus, giving rise to the problem called degradation, for which overfitting is not the reason; rather, it occurs from the time of training. However, the problem of degradation is solved using the deep residual learning that is present in ResNet. ResNet-152 produced the highest accuracy with an error rate of only 3.6%, and because of this, it acquired the first position in ILSVRC 2015. However, no other ResNet architecture has been able to show such accuracy. ResNet-200 registered 5.79% for top 5 error rate in ImageNet 1-crop and 4.93% for top 5 error rate in ImageNet 10-crop. Whereas ResNet-1001 showed an error rate of 4.62% on CIFAR-10, it did not perform well on CIFAR-100 [79, 82].

CNN architecture called Network In Network (NIN) was developed by Lin et al. [83], shown in Figure 3.11. NIN did not have an impressive accuracy as compared to other proposed CNNs. However, the innovative architecture of NIN and its capability of having a better local

abstraction attracted the attention of computer vision society. It has a very simple architecture that consists of three mlpconv (multi-layer perceptron convolution shown in Figure 3.12) layers stacked one after the other and then followed by an average pooling layer. However, like traditional networks, to obtain subsamples a pooling layer can be used between the mlpconv layers. Lin et al. argue that convolution filter is Generalized Linear Model (GLM) which has lower level of abstraction ability. Therefore, GLM is replaced with Multi-Layer Perceptron (MLP) for improving the local model's ability of abstraction [83].

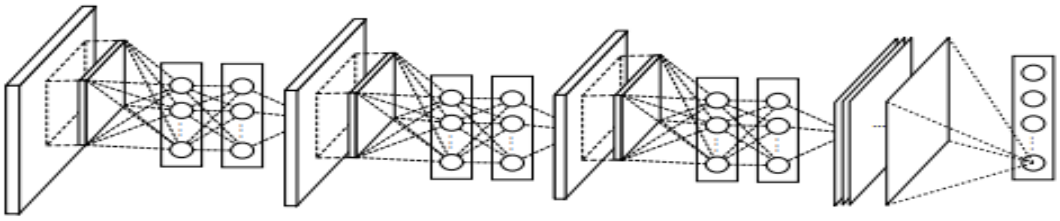


Figure 3.11. Architecture of NIN CNN [83]

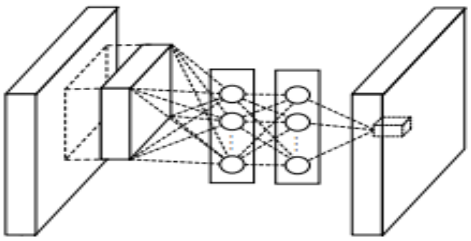


Figure 3.12. Architecture of mlpconv layer [83]

Using average pooling layer, the NIN architecture has also addressed the problem of an increase in the number of parameters in the FC layer (explained in VGG Net) [83]. The use of the average pooling layer also helps to sort out overfitting. The average pooling layer is responsible for creating the feature maps required and then calculates the average of each feature map created. The calculated average is then passed to the softmax layer. Therefore, the advantages of having an average pooling layer are that it helps to not only in keeping the information of the extracted features but also develops the relationship between the features and classification solves the overfitting problem and the spatial conversion of input image is robust as it sums up the spatial information received for all the input images. The well-known universal approximators are the MLP (Multi-Layer Perceptron) and radial basis networks and are used when the distribution is not known and to extract the features' universal function. Also, MLP is a deep structure and can be trained using backpropagation. Furthermore, in NIN architecture Cascaded Cross Channel Pooling (CCCP) is implemented along with a convolutional layer. The CCCP method is equivalent to that of the 1x1 convolutional layer

(i.e. if there are three layers of MLP in mlpconv layer then the output from the convolutional layer goes through three 1x1 convolutional layers having ReLU activation) [83].

The other CNNs developed by keeping Fully Convolutional Network (FCN) [84] as their foundation are FPN (Feature Pyramid Network) [85], U-Net [86] and V-Net [87] on Biomedical image dataset, SegNet [88], LinkNet [89], Fully Convolutional DenseNet [90], PSPNet (Pyramid Scene Parsing Network) [91], RefineNet [92], G-FRNet (Gated Feedback Refinement Network) [93], DecoupledNet [94] and GAN (Generative Adversarial Network) [95]. All these networks are for semantic segmentation and object detection and have registered good performance on datasets like Biomedical images, CamVid [154. 155], Cityscapes [156], MS COCO [102], SUN RGB-D [150, 151], PASCAL VOC [145] and ImageNet [143].

All the discussed neural network architectures have different logics and also have addressed different problems associated with ANNs. Considering the logic used in forming discussed neural network architectures, a new methodology for forming a new neural network architecture becomes feasible.

3.3.3. Fast R-CNN and Faster RCNN

Before understanding Fast and Faster RCNN, it is important to look the R-CNN (Region based Convolutional Neural Network) that first helped to develop the object detection capability in CNN. The object detection using R-CNN generally has two steps to follow, the first is the region proposal, and the second is the classification step. Selective search is the heart of R-CNN as it helps to find different regions that may contain objects. This helps to form the region's proposal, combined as image size and fed into CNN. Then each region's feature vector is extracted by CNN. Then the trained linear SVM (support vector machines) are fed with the region vectors as input. The region vectors are also provided to the bounding boxes regressor so that the exact coordinates are obtained for bounding boxes. Bounding boxes having overlap problems with each other are dealt with using the non-maxima suppression. When the region proposal and classification step is applied to FCN (Fully Convolutional Network), then that architecture becomes R-FCN (Region based Fully Convolutional Network), which is also effectively used for object detection purpose. The mechanism in R-FCN for object detection remains the same as that of R-CNN [96].

Fast RCNN is dependent on external region proposal methods like the selective search [97]. It, along with CNN, has the proposed regions layer and classifier and box regressor layer. In the Fast RCNN architecture shown in Figure 3.13(a), the CNN processes the images first and sends them as an input to the RoI (region of interest) pooling layer. In this layer, the input

image is resized according to the concerned regions. Then the output from the RoI pooling layer is sent to the FC layers as an input, and then using the softmax layer, the possible outcome is found out, and using the bbox (bounding box) regressor, the boxes on the objects are created to point out the features detected in the image by the trained neural network. The Fast RCNN reports that it is nine times faster than R-CNN in training the VGG-16 and is 213 times faster in testing time [97]. On comparing Fast RCNN with SPPnet (Spatial Pyramid Pooling Network) [98], the Fast RCNN reported that it is more accurate [97]. It trains VGG-16 three time faster and is ten times faster in testing [97].

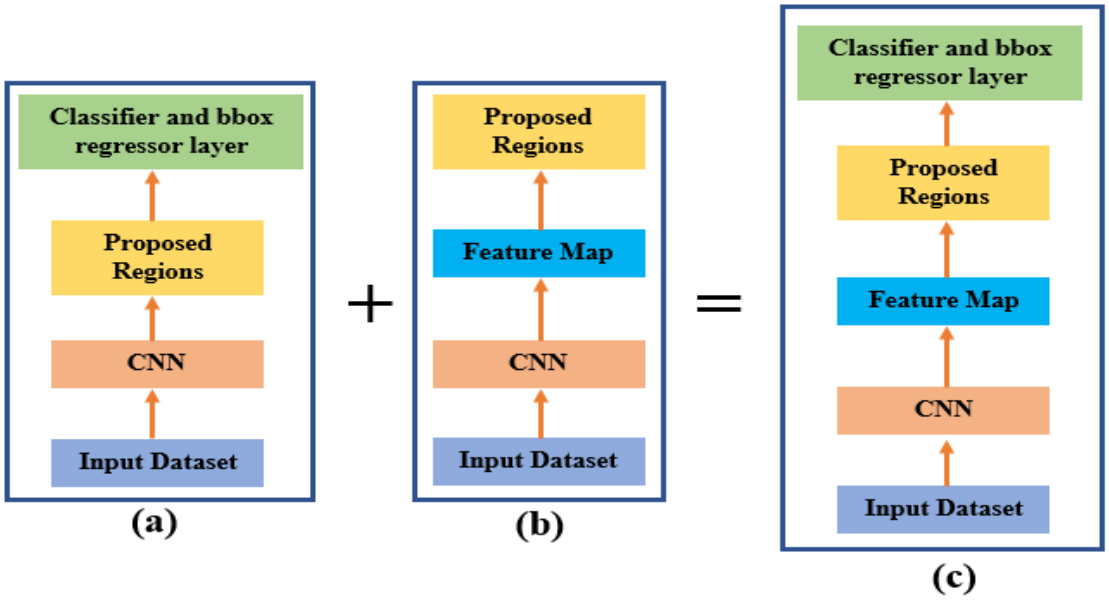


Figure 3.13. Addition of Fast RCNN and RPN to form Faster RCNN, (a) block diagram of the basic architecture of Fast RCNN, (b) Block diagram of the basic architecture of RPN, (c) Block diagram of the basic architecture of Faster RCNN.

RPN (Region Proposal Network) [99], shown in Figure 3.13(b), has CNN, feature map, and proposed regions. In this architecture, the anchor mechanism is used to judge if the particular area on the image contains an object or not. To judge it, the following formula is used,

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ union} \begin{cases} > 0.3\ object \\ < 0.3\ no\ object \end{cases} \quad (5)$$

where,

IoU – Intersection over union

If the value of IoU is greater than 0.3, then that anchor contains the object, and if the IoU value is less than 0.3, then that anchor does not have any object [100]. Ren et al. [100] propose two kinds of anchors with positive labels, (a) anchors with the highest overlap of IoU with the ground-truth box and (b) with any ground-truth-box, the IoU overlap is higher than 0.7. According to [100], proposal (b) is sufficient in determining positive labels. However, the

proposal (a) is adopted because in case of proposal (b), sometimes no positive result could be found. Therefore, IoU with value more than 0.3 is assigned positive anchor.

The Faster RCNN is the amalgamation of Fast RCNN and RPN [100]. In its architecture, shown in Figure 3.13(c), there is a CNN, feature map, proposed regions and classifier, and bbox regressor [100]. Thus, making the neural network better than Fast RCNN by giving it the power of RPN, which Fast R-CNN does not have. Faster RCNN reported speed of 5 fps with a single GPU (Graphics Processing Unit) while training VGG-16 and reported exceptional accuracy for object detection performed on PASCAL VOC 2007, 2012 [101] and MS COCO datasets [102]. They only had 300 proposals per image.

3.3.4. Mask R-CNN

In the existing neural networks, there is the presence of object detection and semantic segmentation. However, the existing neural networks are not capable of instance segmentation. The Mask R-CNN using mask fulfills the criteria of instance segmentation [103]. The Mask R-CNN is the combination of Faster R-CNN and FCN. Fast and Faster R-CNN have classifiers and bbox. However, Mask R-CNN, in addition to these two, has a mask. Mask R-CNN uses RoI Align that helps in storing spatial features without any loss of information.

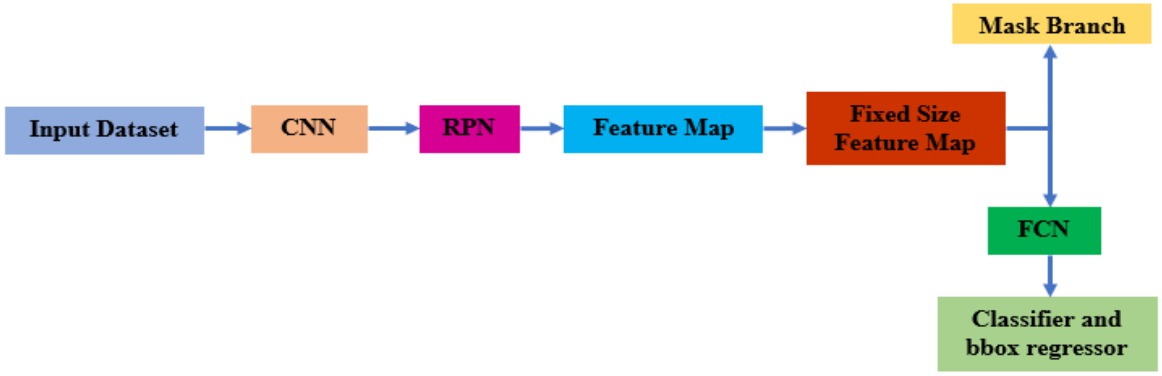


Figure 3.14. The block diagram of the architecture of Mask RCNN that is almost similar to that of Faster RCNN but with an additional feature of Mask Branch that provides the power of instantaneous segmentation.

Figure 3.14 shows the block diagram of the Mask RCNN architecture with the property of instantaneous segmentation until the invention of Mask RCNN lacked in other neural networks. From the block diagram, it is clear that the architecture is very much similar to that of Faster RCNN. However, the only difference between Mask RCNN and Faster RCNN is the mask branch which appears in the Mask RCNN architecture, responsible for providing the neural networks the power of instant segmentation. The architecture of Mask RCNN contains CNN followed by RPN and feature map. The feature map is then connected to the fixed-sized feature map where the RoI Align is also in it. The output from the fixed-size feature map is split and

fed as input to the FCN and mask branch. From the mask branch again, the bounding box regressor helps in object detection. The classifier layer classifies different features learned, and mask branch, on the other hand, performs instantaneous segmentation simultaneously. The architecture reported a speed of 5 fps and was the winner of the COCO challenge 2016. The architecture was trained using the MS COCO dataset for COCO challenge. Mask RCNN is also utilized for estimating the different human poses and has reported a satisfactory result.

3.3.5. Different Scene Recognition Tasks

Sun et al. [104] developed Unified Convolutional Neural Network, shown in Figure 3.15, that can parallelly perform scene and object detection. The dataset used for the task is the SUN RGB-D dataset [150, 151]. The dataset is used for both scenes and object recognition. The input images are parallelly trained on both CNN architecture, i.e., the scene data is trained using Scene Recognition Network (SRN) and object data is trained using RPN (Region Proposal Network) parallelly. The output from the SRN and RPN are fused (concatenated) and sent to a common FC layer. Before undergoing fusion, the information meant for scene information goes through the RoI pooling layer that helps to extract global scene features. The object-related information goes through the regional object features layer. It is then installed in a robot operating system (ROS) for semantic mapping and grasp detection (It helps to accomplish the complex task of manipulation). To accomplish the installation in a robot system, the authors proposed an update system that can convert CNN's predictions to actual robot beliefs. The algorithm that can process point cloud dataset is developed to detect 2-Dimensional images in a 3-Dimensional world. Sun et al. [104] to have developed a system with low latency in the presence of limited resources.

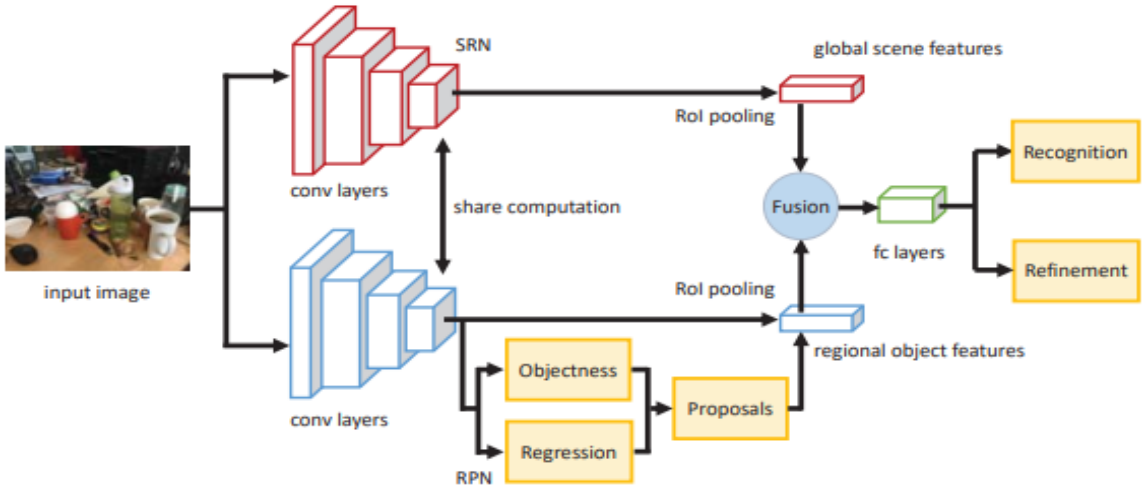


Figure 3.15. Architecture of Unified Convolutional Neural Network [104]

Zhou et al. [105] created a scene dataset that consists of 10 million images categorised under 365 different categories and named it as Places365 dataset. For comparison, a dataset of 2.5 million scenes categorised under 205 different categories is also used which is called Places205 dataset. The whole scene dataset has both internal and external scenes. The Places365 and Places205 datasets are used to train different CNNs and hence after the training are named as Places365 and Places205 AlexNet, Places365 and Places205 GoogLeNet, Places365 and Places205 VGG and Hybrid1365-VGG. Hybrid1365 is the dataset formed using the 1000 classes of ImageNet dataset and 365 classes of Places dataset. Then it is used in training VGG. All the trained CNNs are tested on different datasets like MIT67 [9], SUN397 [147, 148], Scene15 [152], SUN Attribute [149], Caltech 101&256 [157, 158], and Event8 [153]. VGG, on average, produced higher accuracy as compared to AlexNet and GoogLeNet. Therefore, it was also trained using Hybrid1365 [105]. Hybrid1365-VGG produced the highest average accuracy on all datasets used for testing.

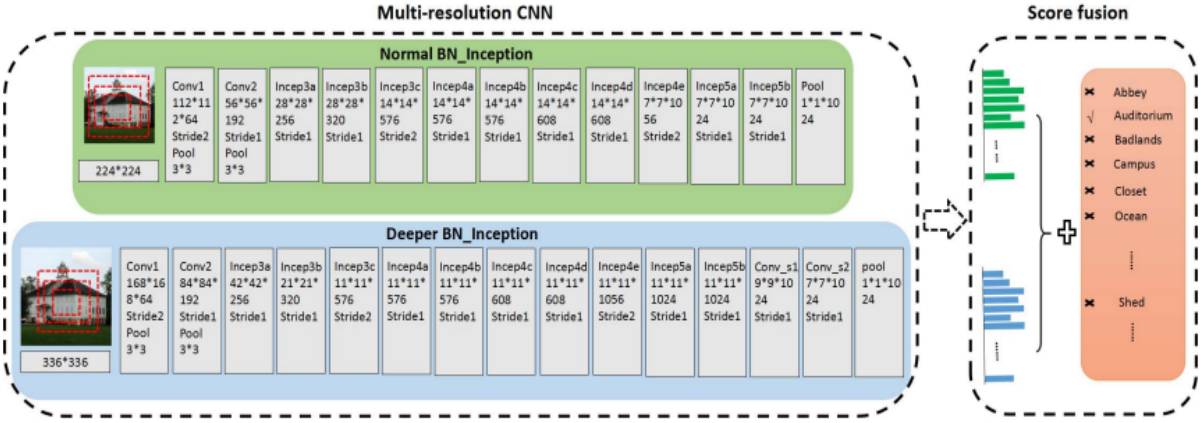


Figure 3.16. Architecture of Multi-Resolution CNN [106]

Wang et al. [106] introduced Multi-Resolution CNN, shown in Figure 3.16. This CNN specifically improves CNN's scene recognition capabilities by addressing the issues like differences in the large intra-class variations (local objects, global layout and background environment present in a scene) present in the dataset and label uncertainty because of the proliferation in the number of categories in the scene. Multi-Resolution CNN consists of coarse and fine resolution such that it can at multiple levels extract content and structure of different regions. To solve label uncertainty, a confusion matrix of validation data is used to create a super category by merging the similar classes and creating soft labels for each image using the extra network. In the Multi-Resolution CNN, Inception with batch normalization is implemented. The inception part is responsible for multi-scale processing. This helps in the proper development of a network that can perform scene recognition. The coarse-resolution

part of Multi-Resolution CNN has the first two layers of convolution with Max Pool which transform input image with dimension 224x224 into feature maps of dimension 28x28. The transform helps in faster processing of information in later layers. Then there are 10 inception layers. At last, it has an average pool layer. For the activation of the convolutional layer, batch normalisation is used, followed by ReLU activation. The fine resolution part has the first two layers of the convolutional layer with Max Pool, just like the coarse part. Then it is followed by ten inception layers, two convolutional layers. However, without Max Pool and average pooling at last.

Afif et al. [115] proposed EfficientNet specifically for indoor scene recognition tasks. Convolutional layers and Mobile Bottleneck Convolution layer (MBConv) combination helps in developing the EfficientNet. Convolutional layers are responsible for feature extraction, whereas to make computation less complex, MBConv encodes the lower-dimensional subspace's feature maps. The transfer learning process is used to train the neural network. The neural network is tested on indoor home scene datasets with bathroom, bedroom, kitchen and living room scenes. The higher accuracy produced by the proposed EfficientNet makes it a state-of-the-art method.

Espinace et al. [107, 108] introduced the concept of performing scene recognition using object detection. The method tries to detect the objects in a scene and using the information produces the scenes' probability. For example, if the system detects a projector screen and a clock, it gives the conference room the highest probability. The concept is applied for detecting only four different scenes, namely, conference room, hall, office and bathroom. The method uses the generative probabilistic hierarchical model. This is done specifically for accomplishing the semantic segmentation. The Monte Carlo sampling method [109] is also used to fit in the improved object classification using real-time 3D range sensors that generate refined geometrical information of objects. This method is not at all based on neural networks and is embedded in a mobile robot.

Scale-Invariant Features Transform (SIFT) [110] and Speed Up Robust Features (SURF) [111] are some other scene recognition techniques that do not involve neural networks. In Li et al. [112] and Sudderth et al. [113], a scene recognition task is performed based on object recognition, which does not involve neural networks. Indoor scene recognition is also done using technique that does not involve neural networks, CodeBookless Model (CLM) [112]. This method reported 20% more accuracy than any other traditional codebook construction. Scene 15 [114] is the dataset used in the CLM method.

3.3.6. 3D Object Detection Methods

Kanezaki et al. [116] propose a new CNN, RotationNet to recognize 3D objects. The dataset used are ModelNet-10 and ModelNet-40 [117]. RotationNet not exactly uses the raw 3D data to learn. However, the images are taken from one or more different angles of a 3D image. Those images are used to train the CNN. Ultimately making the complete process based on 2D. This increases the size of the dataset, which is ideal for CNN to produce higher accuracy. In RotationNet, the dataset so formed after taking images of 3D objects from three different angles becomes larger to that of the original dataset. RotationNet is a further advancement of multi-view CNN (MVCNN) [118]. The working of RotationNet is further shown in Figure 3.17. RotationNet, till now, registered the highest accuracy of ModelNet-10 and ModelNet-40 datasets.

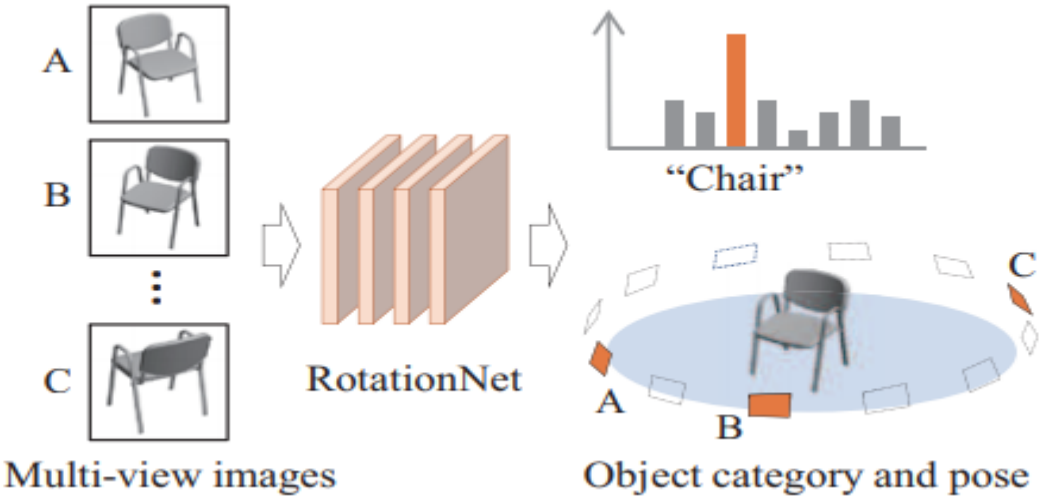


Figure 3.17. RotationNet working [116]

Garcia-Garcia et al. [119] introduced PointNet Convolutional Neural Network (PointNet CNN), shown in Figure 3.18. The developed CNN is to directly use the point cloud dataset instead of converting it first to 3D voxel grids and then training the CNN. The conversion of the point cloud dataset to a 3D voxel grid is required for other neural networks because of the geometrical irregularities present in the point cloud. The geometric irregularities are no proper edges, lines and curves. Therefore, this makes the information extraction difficult from the point cloud dataset. The dataset used for training is ModelNet-10. The CNN has a point cloud data layer where the specifications of each image are specified. The specifications include the voxel grid size and leaf size. Then the architecture has two convolutional layers with Max Pool layers. There are two fully connected layers with a softmax layer at last.

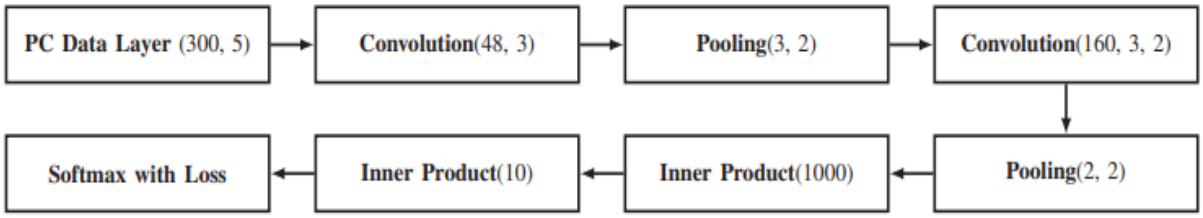


Figure 3.18. PointNet CNN Architecture [119]

Qi et al. [120] introduced PointNet, shown in Figure 3.19. PointNet architecture can be directly trained using the point cloud dataset, just like PointNet CNN. It can recognize the 3D objects even after being trained on a 1D array dataset. The dataset used to train PointNet is ModelNet-40. The number of points specified is taken as input n, during the network's training [120]. The backbone of PointNet is MLP (Multi-Layer Perceptron) [121]. The PointNet has two parts, the first part is the classification network and the second part is the segmentation network. The classification part takes the number of points as input, which is applied with input and feature transformation. The input and feature transform are nothing but the T-Nets shown in Figure 3.19, which helps in predicting affine transform [120]. Then aggregation of data is processed using the Max Pooling layer. For k number of classes, classification scores are generated. After this, the segmentation layer is responsible for concatenating the local features, per point scores of output, and global features. ReLU activation function and batch normalization are used in every layer. The last layer, which is MLP is equipped with dropout.

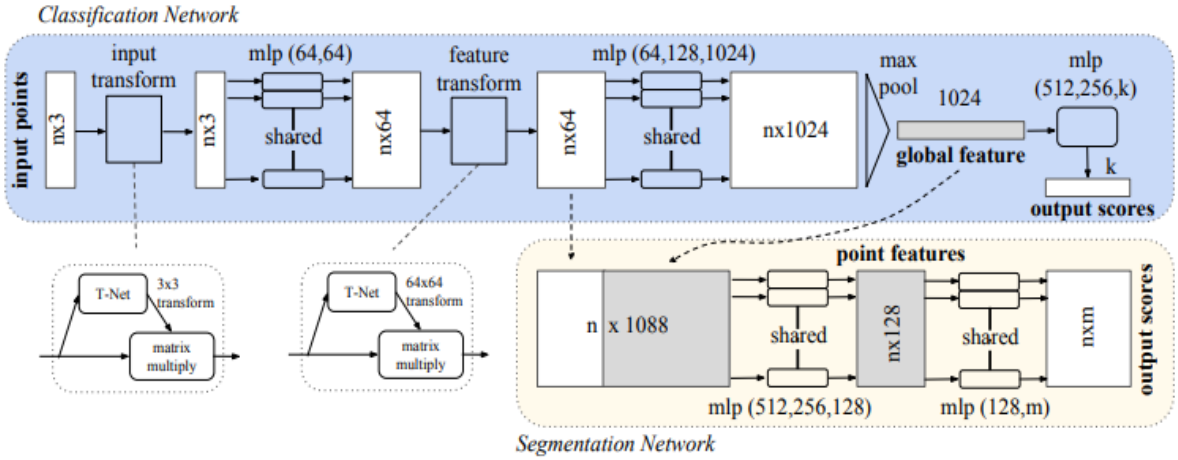


Figure 3.19. PointNet Architecture [120]

Self-Organizing Network (SO-Net), shown in Figure 3.20, developed by Li et al. [122] is also a deep neural network that is directly trained using a point cloud dataset. SO-Net builds a Self-Organising Map (SOM) by forming a model for the point cloud's spatial distribution. Then feature extraction from SOM nodes and individual points is performed. A single feature vector represents the information. In SO-Net, the encoder part normalization of input points with k-

nearest SOM nodes. Later, using the Max Pooling, the normalized resultants are max pooled. The Max Pooling is based on kNN (k-Nearest Neighbour) search (point to node) on SOM. The field overlaps receptive is determined by k. In the segmentation part of SO-Net, following the kNN association technique, the normalized kN points are concatenated with M node features. At last, using the average pooling, the aggregated N features are obtained from kN features.

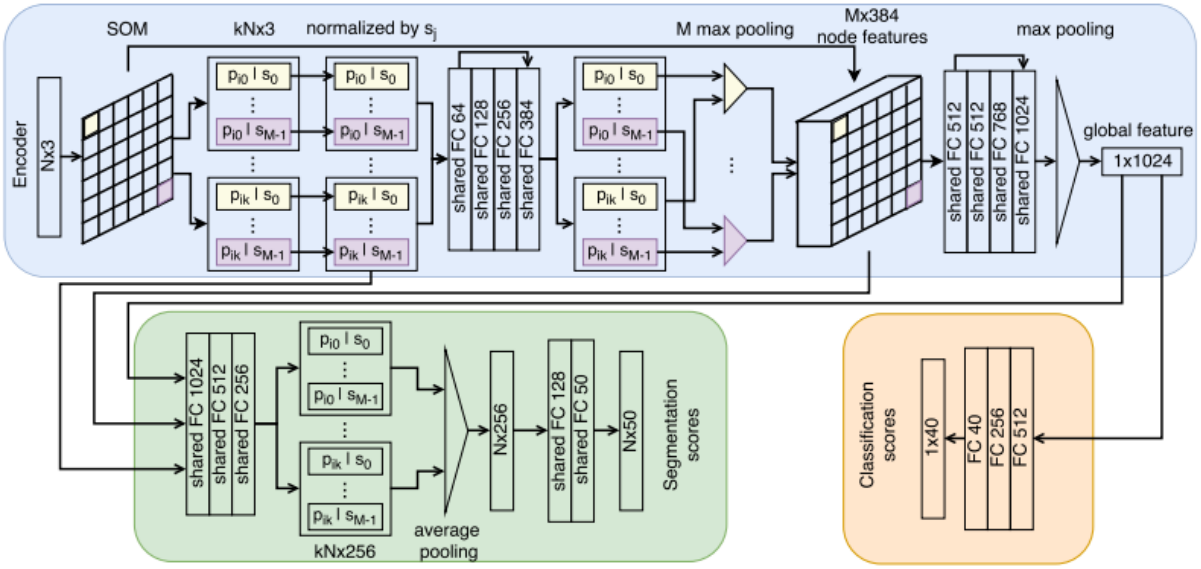


Figure 3.20. SO-Net Architecture [122]

Achlioptas et al. [123] introduced a new architecture of Auto-Encoder (AE) called deep AE. The AE is specifically developed for point cloud dataset. EMD (Earth's Mover Distance) and CD (Chamfer Distance) loss functions are used during AE training. Further, Achlioptas et al. [123] develop two different Generative Adversarial Networks (GAN), namely, raw-GAN (r-GAN) and latent space GAN (l-GAN).

In r-GAN, AE is used as a discriminator which has leaky-ReLU activation function. The last part of r-GAN has 5 fully connected layers with the ReLU activation function, which is fed to a sigmoid function that produces the final result [123]. Whereas in l-GAN, a pre-trained AE (EMD (Earth's Mover Distance) or CD (Chamfer Distance) loss functions are used during AE training on each object class separately) is used from which the data is passed [123]. Further, in l-GAN, a single-layered MLP is used as a generator and a double hidden layered MLP as a discriminator [123].

Separately, a group of GMM (Gaussian Mixture Models) is used for generating point clouds. This is accomplished using fitted distribution sampling and also by using pre-trained AE as a decoder. CD loss function proved to be better for l-GAN as it helped in producing better

accuracy. The performance of GAN improved after being trained in the fixed latent space of AE and GMM.

Apart from aforementioned works, there are some other neural networks that are implemented for the same work like PointNet++ [124], PointConv [125], PointCNN [126], MLVCNN (multi-loop view CNN) [127] and LDGCNN (Linked Dynamic Graph CNN) [128]. They are either trained and tested only on ModelNet-40 or on ModelNet-10 dataset and are known for their accuracy.

3.4. Capsule Neural Network (CapsNet)

Scene and object recognition are currently performed using CNNs. Discriminatively trained multiple layers of feature detectors are used by CNNs. In layers close to input, feature layers' spatial domains get bigger. All these points are advantageous for recognition tasks. However, the subsampling layers that can be interleaved with feature extraction layers help to achieve local invariance by obtaining outputs from the feature detectors nearer to them. This leads to the loss of information on the positions of different things present in an image. Moreover, CNNs do not have the ability to retain the relationship between the extracted features. This leads to confusion when the trained CNN is asked to identify the same object or scene but in different orientations. This is very vital for any developed IAS. Inside home any object can be present in any other orientation. For example, a knife in the kitchen can be found lying on platform, can be found in a knife stand or sometimes even on a vegetable cutter plate. In all the cases the appearance of the knife will be different. For indoor home scenes also, it is applicable. Suppose in a living room, table is kept on the right side and a sofa left side. The IAS learnt that this is living room. After sometime the sofa is moved to right side and table on left side. Still, it remains a living room. An IAS is expected to recognize the knife and the living room correctly in each case. Therefore, to solve this problem, the Capsule Neural Network (CapsNet) is introduced. CapsNet not only focuses on extracting feature of any object but also the orientation. This helps the CapsNet to recognize any object or scene presented to it in any orientation.

Sabour et al. [129] first proposed the idea of CapsNet. To understand CapsNet, first, some basic things are needed to be understood. In computer graphics, the rendering process is followed in which to capture an image, the abstract of that image is first studied (the XY coordinates, angle, etc.), and then using the rendering process, the image is created. Whereas the human brain does just the opposite of what the computer graphics do, and the process is called inverse graphics or inverse rendering, i.e., it first takes the image and then tries to find

out the abstract of that image. In CapsNet, the technique of inverse graphics or inverse rendering is the fundamental principle, and therefore the neural network that performs inverse graphics is a CapsNet. CapsNets are the neural networks with Capsules to perform some complicated and important functions (like affine transformation and squash function) on inputs provided to them. The outputs from the capsules are encapsulated in small vectors.

3.4.1. Basic CapsNet Architecture

The first CapsNet with dynamic routing was proposed by Sabour et al. [129], which was trained on the MNIST dataset. Figure 3.21 shows the basic architecture of CapsNet. The architecture of the CapsNet has six layers,

i. The Convolutional layer
ii. The PrimaryCaps (Primary Capsule) layer
iii. The ClassCaps (Class Capsule) layer
iv. Fully connected layer 1
v. Fully connected layer 2
vi. Fully connected layer 3

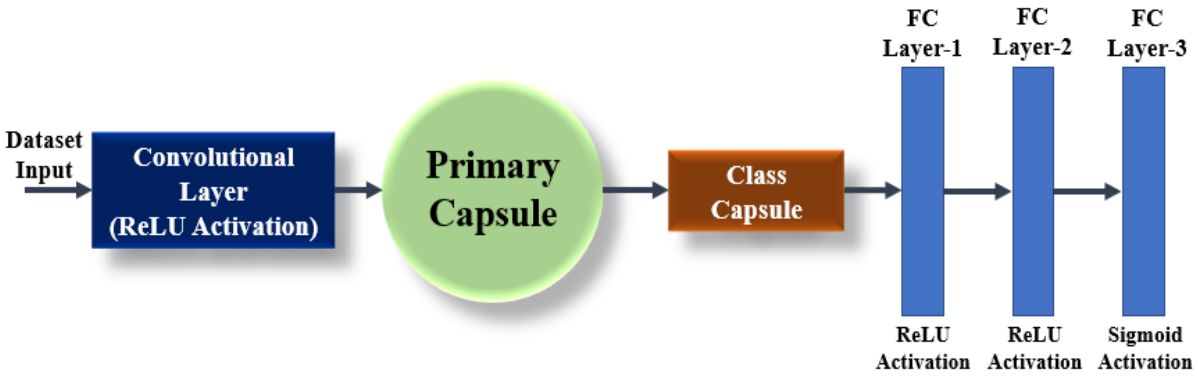


Figure 3.21. The basic block diagram of Capsule Neural Network (CapsNet)

The first three layers (convolutional layer, PrimaryCaps and ClassCaps) of the architecture are categorized under as encoder part and the rest three layers (FC layers) are the decoder part. The convolutional layer helps to extract the important features from the two-dimensional images. This process may involve blurring, embossing, outlining, and sharpening the image. This layer is also equipped with a ReLU activation function, ReLU function is max(0,x) for all values minus infinity to infinity. The PrimaryCaps layer is the combination of capsules present in the architecture which receives the extracted features from the convolutional layer and tries to learn the orientation of the features. The PrimaryCaps layer is able to do it by using the affine transformation, weighted sum and squash function. Further, in PrimaryCaps layer, output is in

vector form that contains probability of the presence of an entity and also the set of instantiation parameter [142]. The PrimaryCaps are further discussed in detail in Chapter 4, section 4.2.3.

The PrimaryCaps are connected with ClassCaps using dynamic routing (Chapter 4, section 4.2.4). The number of ClassCaps is decided as per the number of classes the dataset used for training has. In case of MNIST dataset, there are 10 classes. Therefore, the number of ClassCaps were 10. The orientation of features learnt by PrimaryCaps are sent to the respective ClassCaps. An agreement is reached between the PrimaryCaps and the ClassCaps using dynamic routing that the feature belongs to a particular class of ClassCaps, this is also called routing by agreement (Chapter 5, section 5.3, Figure 5.5). For example, in case of MNIST dataset, the 10 ClassCaps are from 0 to 9. So, if the PrimaryCaps reach an agreement with ClassCaps that the learnt feature is of digit 9, then the features will be sent to ClassCap responsible for taking the features of digit 9. Then it is followed by fully connected layer 1, layer 2, and layer 3. The fully connected layers can be used with a choice of the activation function.

The loss occurring in the CapsNet is calculated using the following mathematical equation,

$$L_c = T_c \max(0, m^+ - ||v_c||)^2 + \lambda (1 - T_c) \max(0, ||v_c|| - m^-)^2 \quad (4)$$

Where,

L_c – Loss term for each ClassCaps.

T_c - Loss function of target ClassCaps class/layer, c – class capsule class/layer.

Λ – coefficient used for numerical stability, and its value are fixed at 0.5.

$T_c \max(0, m^+ - ||v_c||)^2$ calculates the loss for correct digitcap, i.e., when , T_c is 1. $\lambda (1 - T_c) \max(0, ||v_c|| - m^-)^2$ calculates the loss for incorrect digitcap, i.e., when , T_c is 0. The values of λ, m^+ and m^- are 0.5, 0.9 and 0.1 respectively [129].

3.4.2. Different CapsNets (Capsule Neural Networks)

Lambert et al. [130] performed an investigative analysis on traditional CapsNet's performance of ModelNet-10 and ModelNet-40 datasets. For this purpose, the traditional CapsNet is completely converted into a 3D neural network. The datasets are used in 3D voxel grid form instead of raw point cloud dataset. Lambert et al. [130] also focused on how much CapsNet is efficient in retrieving a 3D model. The accuracies produced by 3D-CapsNet on ModelNet-10 and ModelNet-40 datasets, are 93.08% and 82.73% respectively.

Ahmad et al. [131] developed 5 different CapsNets. All the CapsNets were trained on ModelNet-10 and ModelNet-40 datasets in the form of 3D voxel grids. All the CapsNet

architectures are of 3D. Architecture 1, shown in Figure 3.21, has two convolutional layers, primary capsule, digit capsule and two FC layers. The first convolutional layer has leaky ReLU activation function with batch normalization. Second convolutional layer has Max Pooling with batch normalization. Architecture 2, shown in Figure 3.22, has one convolutional layer with batch normalization and leaky ReLU activation function. Further it has primary capsule, digit capsule and two FC layers. Architecture 3, shown in Figure 3.23, is similar to that of Architecture 1. However, in Architecture 3, second convolutional layer has batch normalization with leaky ReLU instead of Max Pooling. Architecture 4, shown in Figure 3.24, is similar to Architecture 2 but it has only one FC layer unlike Architecture 2. Architecture 5, shown in Figure 3.25, has one convolutional layer with batch normalization and leaky ReLU activation function followed by two primary capsule layers. Then there is a digit capsule layer and two FC layers. Architecture 1 produced the highest accuracy of 91.48% on ModelNet-10 dataset and Architecture 2 produced the highest accuracy of 89.66% on ModelNet-40 dataset.

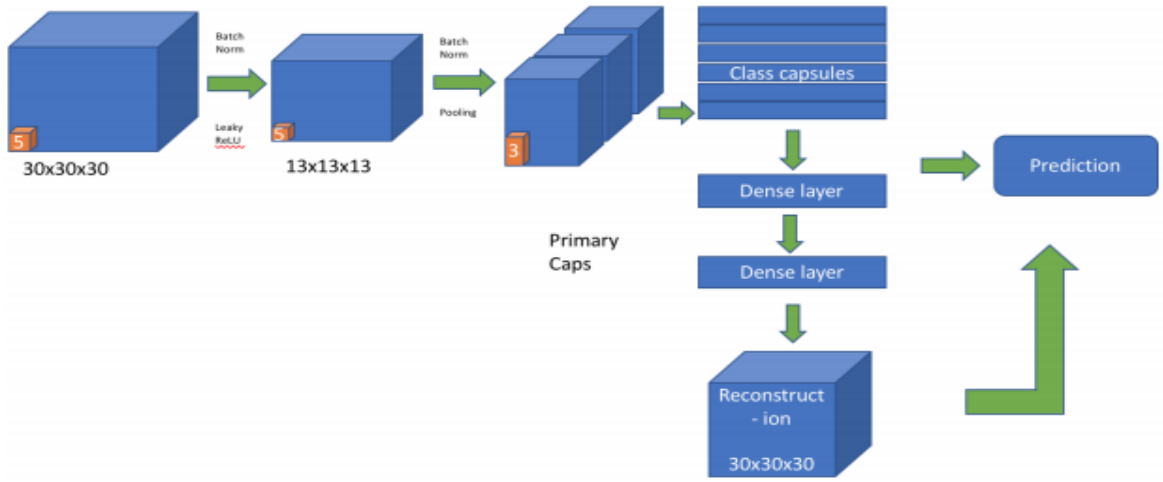


Figure 3.21. CapsNet Architecture 1 [131]

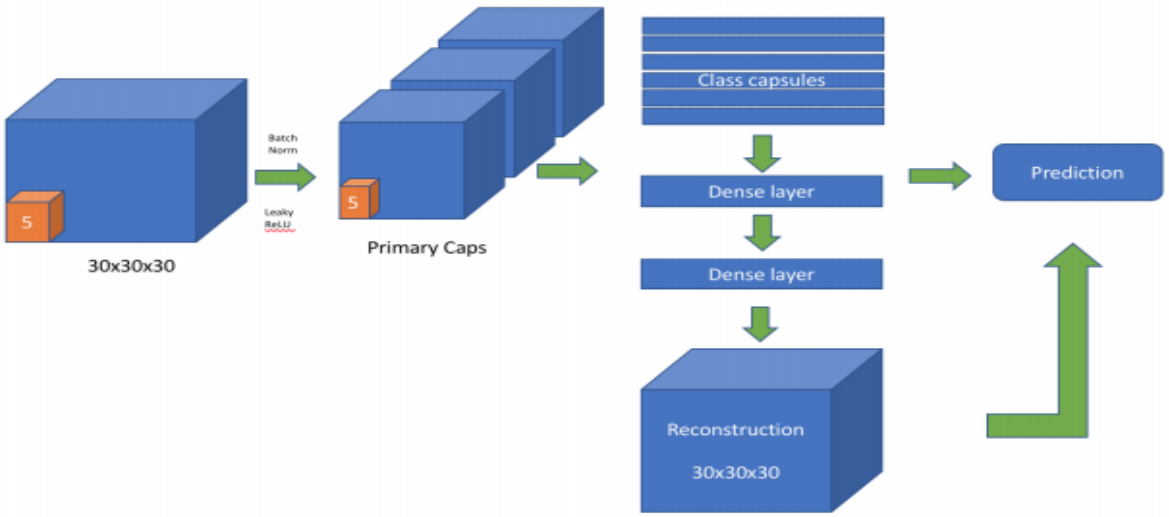


Figure 3.22. CapsNet Architecture 2 [131]

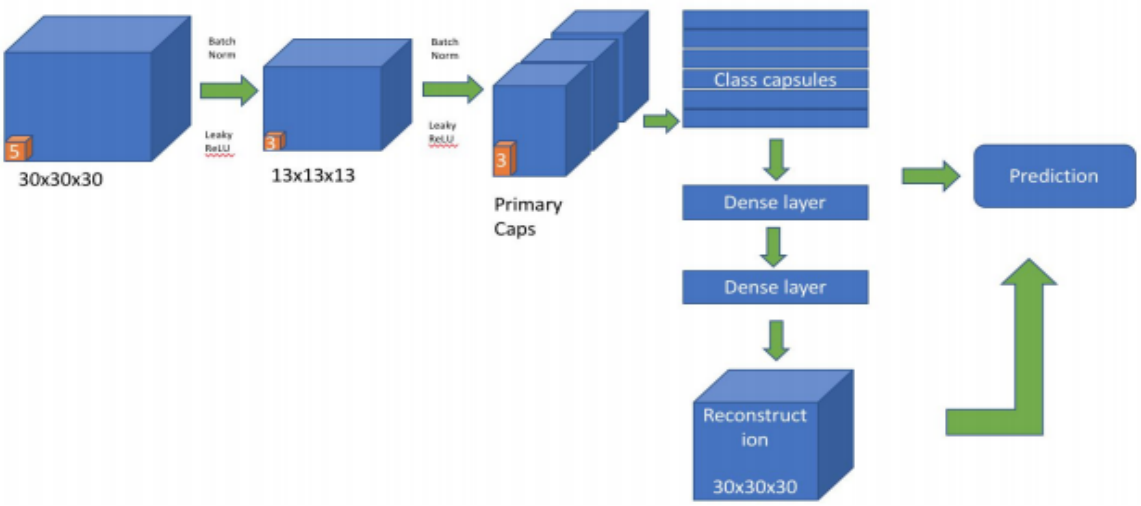


Figure 3.23. CapsNet Architecture 3 [131]

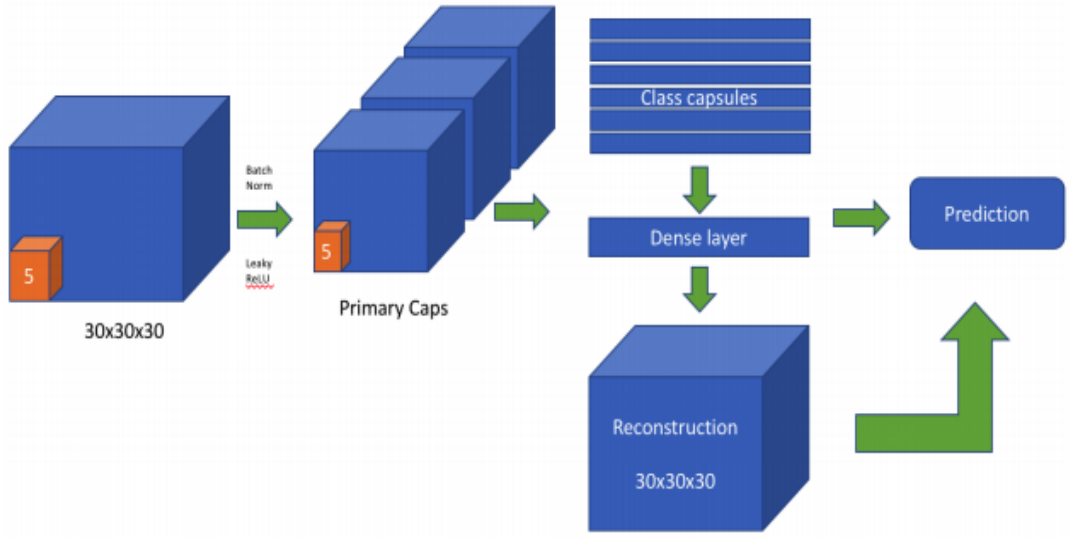


Figure 3.24. CapsNet Architecture 4 [131]

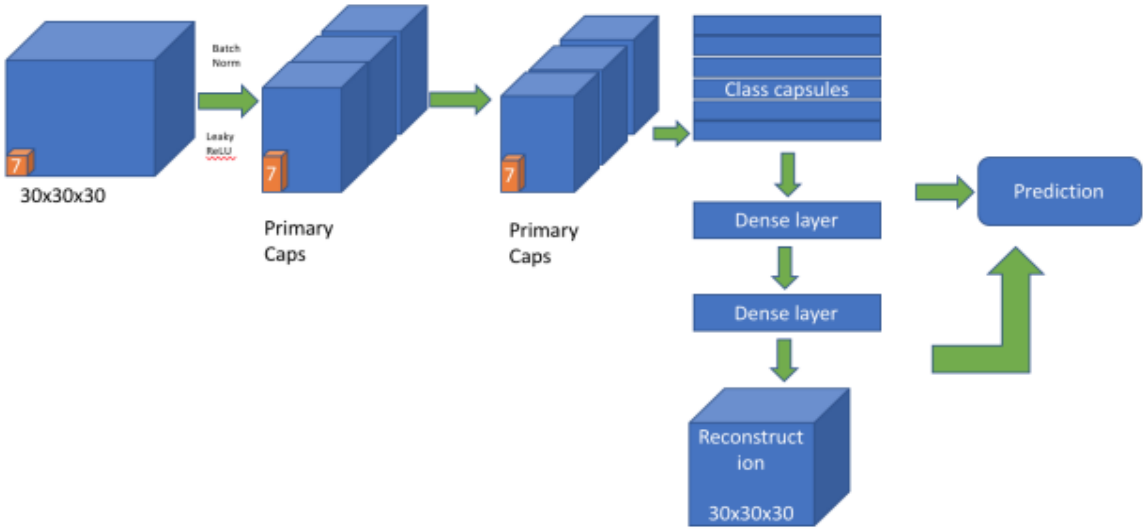


Figure 3.25. CapsNet Architecture 5 [131]

Cheraghian et al. [132] developed a CapsNet in 3D which can be directly trained on point cloud dataset. The CapsNet architecture is inspired from PointNet architecture. The capsule used in the architecture is of 3D. The architecture has feature extraction layer, aggregation layer, feature vector, 3D primary capsule and digit capsule and at last the reconstruction layer which is subsequently attached with a reconstruction loss function. The accuracy produced by the proposed architecture on ModelNet-10 and ModelNet-40 datasets are 94.7% and 92.7% respectively.

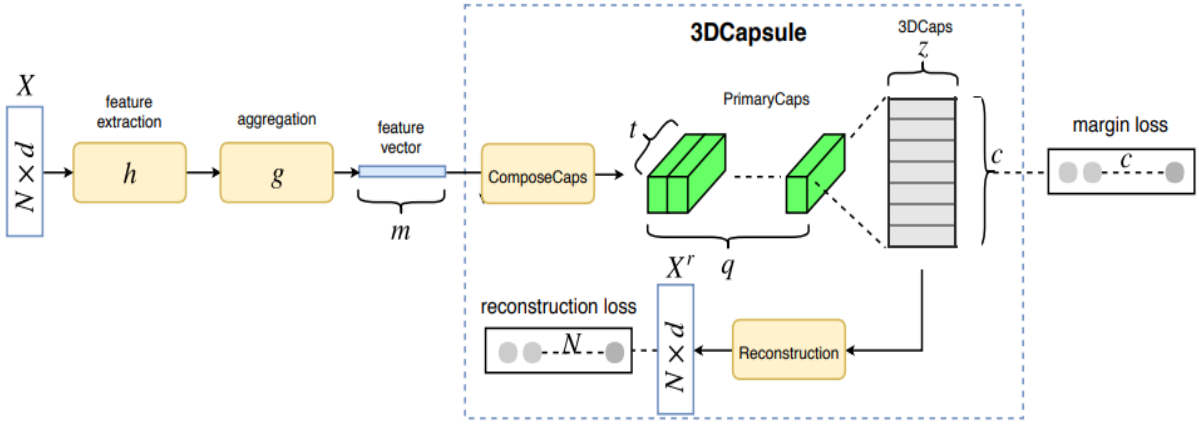


Figure 3.26. 3D Capsule Architecture [132]

3.5. Dataset Summarisation

In this section, the different datasets mentioned in the previous sections of this chapter are summarised in Table 3.1, 3.2 and 3.3. The Tables also summarise the datasets that are used to accomplish the research in this thesis.

Table 3.1. Different datasets used for image classification

Dataset Name	Type	Number of Images
ImageNet [143]	2D RGB Image data with 5247 classes	3.2 million images
ILSVRC [144]	2D RGB Image data with 1000 classes	1.4 million images
PASCAL VOC [145]	2D RGB Image data with 20 classes	11530 images (27450 annotated objects and 6929 segmentations)
Caltech 101 [157]	2D RGB Image data with 101 classes	9146 images
Caltech 256 [158]	2D RGB Image data with 256 classes	30607 images

Table 3.2. Different datasets used for object recognition

Dataset Name	Type	Number of Images
ModelNet-10 [117]	3D RGB Object data with 10 classes	4907 images
ModelNet-40 [117]	3D RGB Object data with 40 classes	12432 images
ADE20K/SUN Database [146]	2D RGB annotated objects with 3169 classes	25210 images
MS COCO [102]	2D RGB annotated objects with 80 classes	330K images with 1.5 million object instances

Table 3.3. Different datasets used for scene recognition

Dataset Name	Type	Number of Images
Places365 [105]	2D RGB scenes with 365 classes	10 million images
Places205 [105]	2D RGB scenes with 205 classes	2.5 million images
Hybrid1365 [105]	2D RGB scenes with 1365 classes	11 million images
MIT67 [9]	2D RGB Indoor scenes with 67 classes	15620 images
SUN397 [147, 148]	2D RGB scenes with 397 classes	108573 images
SUN Attribute [149]	2D RGB scenes with 700 classes	14000 images
SUN RGB-D [150, 151]	2D and 3D RGB-D scene (47 classes) and object (800 classes)	10335 images
Scene 15 [152]	2D RGB scenes with 15 classes	4485 images
Event 8 [153]	2D RGB scenes with 8 classes	1579 images
CamVid [154, 155]	RGB scene videos with object class semantic labels (32 classes)	-
Cityscapes [156]	2D RGB scenes with 30 classes	25000 images

3.6. Conclusion

In this chapter, CNN and its basic concepts were discussed. From the information provided on CNNs, it is clear that for many reasons, as, for example, reducing the total trainable parameters, CNNs prove to be the best choice. They have good performance on tasks, especially related to recognizing objects. However, there are many serious problems associated with CNNs. One of the most common problems is the requirement of very large datasets. Large datasets for every purpose are not always readily available. The other problem associated with CNN is its inability to extract the information regarding the orientation of different objects.

This chapter also discussed all the aspects of CapsNet and its components. The description given on CapsNet helps to understand that it overcomes many disadvantages present in CNNs. The biggest problem that it solves is the loss of orientation information of any image. Routing by agreement present in a dynamic routing algorithm helps in proper information transfer between neurons. The most crucial aspect of CapsNet is that it is capable of recognizing entities from any viewpoint. CapsNet also focuses on properly capturing an entity's complete orientation instead of only focusing on its shape. Therefore, these properties also make CapsNet perform well even on smaller datasets. All these properties make CapsNet one of the most capable neural networks for accomplishing the aims mentioned in this thesis.

CapsNet properties theoretically are the best option for implementing an object recognition system in an indoor environment. However, all the discussed properties need practical

verification. Therefore, the same is done by implementing CapsNet in different ways to make the tasks possible in this thesis.

Nonetheless, the total number of trainable parameters in CapsNets is always very large, and this is an issue which needs to be addressed. Higher total number of trainable parameters in neural networks complicates the neural network's training process. This adversely effects the training speed of the neural networks. To speed up learning for large total number of parameters, higher hardware resources are required which cannot always be readily available. In case of three-dimensional datasets, training of a neural network becomes very difficult because of even higher total number of trainable parameters. However, a 3D dataset is generally very small because the representation of each object from different angles is not required. Some research do use three-dimensional datasets, but the trainable parameters of neural networks are too many. Unlike three-dimensional datasets, two-dimensional datasets usually require large number of images because, in such cases, several images of a single object are needed depicting different angles/views in order to train the neural networks adequately. However, there are not many databases available with such large numbers of indoor images. Alternatively, neural networks which can train accurately with relatively small size datasets could provide solutions to assist indoor navigation and object location. Furthermore, additional problems exist in different deep neural networks used till now for scene and object recognition, and they are:

1. All the state-of-the-art CNNs developed along with different functions like batch normalization and inception are trained on large datasets. Therefore, there are no neural networks that can offer state-of-the-art performance on smaller datasets like those existing for indoor scenes and objects.
2. There are no neural networks specifically trained and explicitly tested on indoor home scene recognition. All the neural networks implemented for scene recognition are trained on mixed scene categories (that involves both internal and external scenes).
3. The best performing neural networks have a very complicated architecture. For example, ResNet-152 [79] which has 152 layers along with techniques like batch normalisation and inception. This becomes very complicated for those working with limited resources or on a very small scale.
4. There are research related to scene recognition using object detection. However, they are not implemented using neural networks. It can be said that such research are partially an example of expert systems.

Different CapsNets discussed do have good accuracy but are not efficient because of higher trainable parameters. Moreover, CapsNets have not yet been deployed for scene recognition tasks. In the next chapter, CapsNets for indoor home scene recognition will be implemented and their performance results will be discussed. A new CapsNet, named NoSquashCapsNet is also developed for performing indoor home scene recognition.

Chapter - 4

Traditional CapsNet and NoSquashCapsNet for Indoor Home Scene Recognition

4.1. Introduction

In this chapter, a traditional Capsule Neural Network (CapsNet) is implemented for indoor home scene recognition. This implementation is done for the first time as traditional CapsNet has never been used for this purpose before. This is done because the orientation of objects in indoor scenes is very important and must be properly captured. The aim therefore is to examine the suitability of CapsNet, which is already known to provide object orientation advantages, to be used for indoor home scenes recognition and thus provide a unique tool for IAS systems. Outdoor scene and interior scene classification (not necessarily indoor home scene) tasks have been done using many Convolutional Neural Networks (CNNs). However, since there is an absence of large home scene datasets, there is no CNN that is trained, validated and tested only on indoor home scenes.

Nonetheless, it is important to train systems that are capable of working within a home environment, using correct relevant data. Ideally then a different type of neural network which is not very deep and can achieve comparable accuracy to CNNs using smaller datasets is required.

Further in this chapter, a new CapsNet structure (NoSquashCapsNet) which has capsules but no squash function and also has Max Pool layers is proposed to perform indoor home scene recognition to mitigate the problem of high total number of trainable parameters associated with traditional CapsNet. Hence, the reasons behind developing this new network are to: a) achieve higher accuracy on smaller datasets since, as already mentioned, there is a lack of larger indoor home scene datasets and b) to reduce the total number of trainable parameters. For training and testing, 20000 and 5000 images were used respectively. The dimensions of the images are 128x128. The images were also converted to grayscale.

In this chapter, therefore, section 4.2 is about the traditional CapsNet's implementation for indoor home scene recognition. This section briefly examines the CapsNet and the capsule.

Further, it examines the basic architecture of a capsule, looks at the dynamic routing algorithm which is used to connect the capsules, discusses the process of implementation and also discusses various results obtained from traditional CapsNet. Section 4.3 is about the novel NoSquashCapsNet for indoor home scene recognition. This section explains the complete development and implementation of NoSquashCapsNet. The result produced by NoSquashCapsNet is also discussed. Section 4.4 concludes the chapter.

4.2. Traditional CapsNet for Indoor Home Scene Recognition

Various scene recognition tasks carried out so far have already been discussed in Chapter 3, subsection 3.3.5. These scene recognition tasks are not entirely trained and tested on indoor home scene data. This becomes problematic for IAS developed explicitly for indoor home tasks. An IAS, with no proper ability to recognise indoor home scenes, is of no particular use since they may not be able to assist the person staying indoors.

Currently there are very few datasets with indoor home-specific images available. An indoor home-specific scene set for living room, dining room, kitchen, bathroom, and bedroom is the Places365 dataset [105] which was used in this research. Each of the scenes has 5000 images. Hence, the total number of images in this dataset is 25000. Out of 25000 images, 20000 images are used for training (4000 from each scene) and 5000 for testing (1000 from each scene). However, this is significantly less when compared to other available datasets to train neural networks and thus produce better accuracies. The indoor home scene dataset is also used on Mask RCNN, Fast RCNN, and Faster RCNN. This also helps in analysing the performance of CNNs on smaller datasets and compare them to CapsNet. Furthermore, all the neural networks' performances are also analysed on a reduced indoor home scene dataset. The reduced dataset size was 5000 images for training and 1250 for testing, evenly distributed over the 5 categories or scenes.

4.2.1. CapsNet

Figure 4.1 shows the CapsNet architecture proposed by Sabour et al. [125]. The basic CapsNet architecture has already been discussed in Chapter 3, subsection 3.4.1. The first layer of the CapsNet has one convolutional layer with ReLU activation function. This is the same as that in CNN, which has been explained in Chapter 3, section 3.3.1. The second layer is the primary capsule (PrimaryCaps) which is the backbone of any CapsNet. The primary capsule is responsible for extracting the spatial relationships. Before the information from the convolutional layer is fed to primary capsule layer, the information is first converted from

matrix to vector. This is done because capsules accept vector inputs and produce vector outputs. The next layer, i.e., Class/Digit Capsule (ClassCaps/DigitCaps) takes the information from a primary capsule using the dynamic routing method. Each DigitCaps is responsible for accumulating the information of each class. Finally, the fully connected (FC) layers work as a decoder, i.e. they try to reconstruct the image from acquired information. The final FC layer has a sigmoid activation function so that the output probabilities for each class is produced. The class having highest probability is the required output.

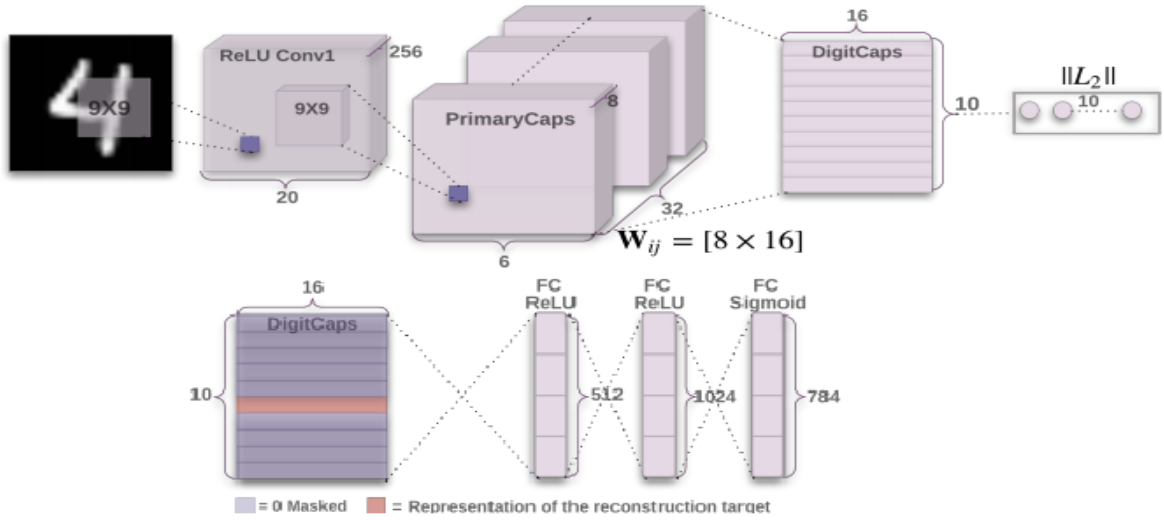


Figure 4.1. Capsule Neural Network (CapsNet) architecture [129]

4.2.2. Capsules: The backbone of CapsNets

Geoffrey Hinton first proposed Capsules [133]. Capsules are responsible for extracting different entities' instantiation parameters, along with their presence. An object or part of an object is detected by a capsule, termed as a visual pathway. There are two types of outputs produced by a capsule,

1. Probability of the presence of a particular kind of object.
2. Orientation, scale, elongation, shear, and position in the receptive field.

There cannot be more than one entity representation in a capsule's receptive field. Therefore, the binding problem (problem of segregating features especially which are of same image) gets solved. Capsules capture similar entities' different properties. This is because of the presence of only one entity at a time. So, when there are identical entities, there are very few chances of having wrong perceptions. This prevents the phenomenon of crowding, which happens when flankers are placed closed to a recognisable object. For recognizing things from different viewpoints, one of the approaches is having extensive data for training the neural network, which is generally practiced in CNN. Another approach is taking images of the same shape and

then change its viewpoint while recognizing it. When the image is shown from a different viewpoint while training, then just averaging the same image from different viewpoints cannot provide an intermediate viewpoint. This is because of non-linearity. If the two images (same image from different viewpoints) are averaged, then two new images are different.

To recognize the image from a different viewpoint, linear space (vector space) with linear images (raw format) is required. Linear space coordinates of identified features' space are acquired by transforming feature space to linear space. This can be done through the following equations,

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{1}$$

Where, M is a matrix that represent feature space where a, b, c and d are the numbers that represent the features. The size of the matrix depends on the size of the filter. This has to be transformed to linear space T_M. Now,

$$T_M(v) = Mv \tag{2}$$

Where, v is a vector, v represents the position vector of point (x, y).

Therefore,

$$T_M(v) = T_M\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = M\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} \tag{3}$$

Equation (3) can also be written as,

$$T_M(x, y) = (ax + by, cx + dy) \tag{4}$$

The other forms of transformation are, rotation, scaling, identity, reflection and shearing [159]. Moreover, it must be clear that blending images (mixing similar images taken from different viewpoints) will not make any difference to the learning process. Instead, the mixing of coordinates must be done by recognizing the relevant parts of images. The complete mathematical process presented is done before the information enters the capsule in vector form. Therefore, from different viewpoints, extrapolation of the recognition of shapes is possible using a capsule.

4.2.3. Basic Capsule Architecture

Figure 4.2 shows the basic working of a capsule [129]. Capsules in the CapsNet are the key for inverse graphics and are inspired by mini-column. They are nothing but functions that help in predicting the different parameters of an image or object. A capsule always takes inputs in

vector form and produces outputs in vector form. It does three basic functions to capture an image; the first function is the affine transformation responsible for capturing the points, straight lines, and planes. The four different types of affine transformations are translation, scale, shear, and rotation.

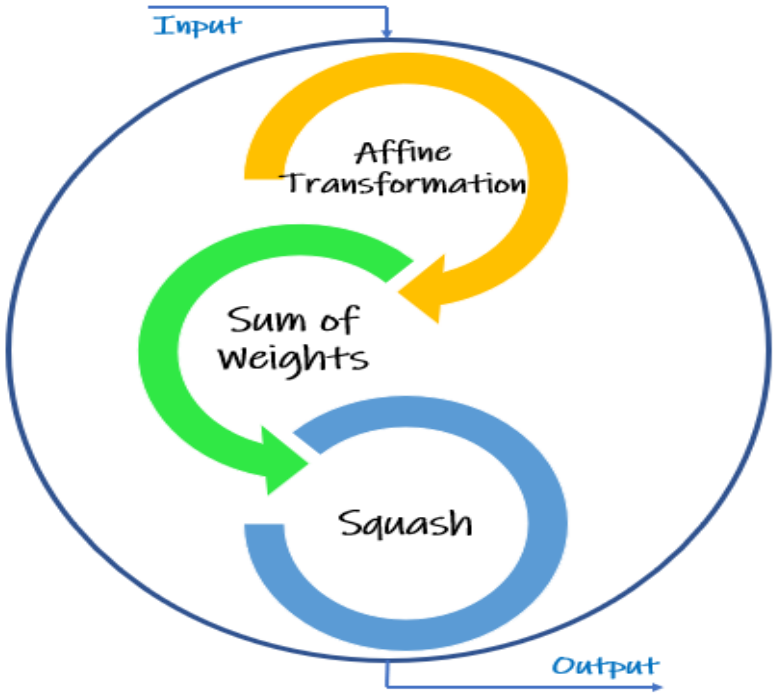


Figure 4.2. Schematic diagram of a capsule

Using these four types, the images' distortions can be easily rectified, and their orientation can be made more explicit. The second function is the sum of weighted input vectors. The combination of inputs is represented in vector form, and the vector addition is performed. The third function is the non-linear activation where the squashing and scaling of the captured image are done. The length does not exceed one, and the direction of the image's orientation remains the same or unchanged because the direction of the vectors remains undisturbed.

The mathematical formulae for the processes involved in the capsule is as follows [135, 136],

For affine transformation,

$$\hat{u}_{i|j} = W_{ij}\, u_i \tag{5}$$

where,

- $\hat{u}_{i|j}$ – Prediction vectors
- W_{ij} – weight matrix
- u_i – output of a convolutional layer

For Sum of weights,

$$s_j = \sum_i c_{ij}\, \hat{u}_{j|i} \tag{6}$$

where,

- c_{ij} – Coupling coefficient, $c_{ij} = exp(b_{ij})/\sum_k exp(b_{ik})$ (7)

In equation (7), $exp(b_{ij})$ and $exp(b_{ik})$ are the standard exponential function of input vector, where k in $b_{ik}$is the number of classes in multi-class classifier. Each c_{ij} is a non-negative scalar. When all c_{ij} are summed up, then the resultant is 1 for each capsule i at lower-level. The number of higher-level capsules is equal to the number of c_{ij} for each capsule i at lower-level. The iteration process of dynamic routing determines the value of c_{ij}.

For Non-Linear Activation (Squashing),

$$v_j = \frac{||s_j||^2}{1+||s_j||^2} \quad \frac{s_j}{||s_j||} \tag{8}$$

Where,

- v_j - output vector of capsule j.
- s_j – total input of capsule j.
- $\frac{||s_j||^2}{1+||s_j||^2}$ – squashing and $\frac{s_j}{||s_j||}$ - unit scaling

4.2.4. Dynamic Routing in Capsules

Dynamic routing in capsules is used to establish a connection between lower-level capsules and higher-level capsules. This helps to develop the capability of generalization in neural networks and the concept of routing by agreement. Routing by agreement means that the higher-level capsule must agree with the output of the lower-level capsule i.e., the lower-level features (like fingers, eyes and mouth) present in lower-level capsules must match the higher-level features (like face and hand) present in higher level capsules. Only then the output becomes the input for the higher-level capsule [129].

Moreover, dynamic routing in capsules helps in the effective use of the information available in capsules. Therefore, the data organization in capsules is in vector form such that the existing probabilities and properties of entities are represented by the length and orientation of capsule neurons. Dynamic routing also helps to find the accurate coupling coefficient (equation 7) by adjusting it between the higher-level capsule and predictive vector. This helps in establishing proper communication between the capsules. Coupling does not directly encode the image; instead encodes the entities present in the image. This makes it clearer for a neural network what exactly it needs to learn [134].

Algorithm 4.1. Dynamic Routing Procedure Steps [129]

Dynamic Routing Algorithm	
1: **procedure** Routing ($\widehat{u_{i\|j}}, r, l$)	
2:	for all capsule *i* in layer *l* and capsule *j* in layer (*l* + 1): $b_{ij} \leftarrow 0$
3:	**for** *r* iterations **do**
4:	for all capsule *i* in layer *l*: $c_i \leftarrow$ softmax (b_i) softmax computes Eq. 7
5:	for all capsule *j* in layer (*l* + 1): $s_j \leftarrow \sum_i c_{ij} \widehat{\boldsymbol{u}_{i\|j}}$
6:	for all capsule *j* in layer (*l* + 1): $v_j \leftarrow$ squash (s_j) squash computes Eq. 8
7:	for all capsule *i* in layer *l* and capsule *j* in layer (*l* + 1): $b_{ij} \leftarrow b_{ij} + \widehat{u_{i\|j}}.\, v_j$
	return v_j

Algorithm 4.1 indicates how dynamic routing functions. From line 1 and line 7, it is clear that the procedure calculates the neural network's forward pass. In line 1, the output vectors $\widehat{u_{i|j}}$ (which are inputs for the capsule), the number of routings *r* and all capsules at lower-level *l* have to be considered before starting the procedure.

Line 2 is to initialize b_{ij} as zero before starting the iteration. In Algorithm 4.1, b_{ij} is a temporary value that is updated iteratively; c_{ij} is responsible for storing the final value of b_{ij} after iteration is over. Line 3 indicates that line 4 to 7 have to be repeated *r* times, where *r* is the number of routings, which is generally three as it is considered ideal to avoid overfitting problems [129]. Line 4 involves calculating c_i, a vector representing all the routing weights of lower-level capsule *i*. Softmax is used to keep c_{ij} positive and also to make its nature probabilistic.

Line 7 does weight updating. As per the formula $b_{ij} \leftarrow b_{ij} + \widehat{u_{i|j}}.\, v_j$, each input is examined, and b_{ij}, which is the corresponding weight, is also updated. This is done by considering capsule *j,* the higher-level capsules. The formula $b_{ij} \leftarrow b_{ij} + \widehat{u_{i|j}}.\, v_j$, indicates that the new value of b_{ij} is the sum of the old value of b_{ij} and the dot product of input of capsule *j* and the output of capsule *i.* The dot product examines the input and output similarity of the capsule. After *r* number of iterations, the outputs from higher-level capsules are produced, and routing weights are also updated. This is then used by the next subsequent layer of the neural network.

4.2.5 Traditional CapsNet architecture

The indoor home scene images in the dataset (indoor home scene extracted from Places365 dataset [105]) are of dimension 256x256. All the images are of RGB scale. So, the depth of the

images is 3. Since using a dimension of 256x256 leads to a very high total number (approx. 200 million+) of trainable parameters, the images are converted to a 128x128 dimension. The conversion is performed using the shrinking process (a group of pixels which are spatially adjacent are mapped as one pixel) in Digital Image Processing [159]. The conversion of dimension reduces the total number of trainable parameters, which further helps in the faster training process of neural networks.

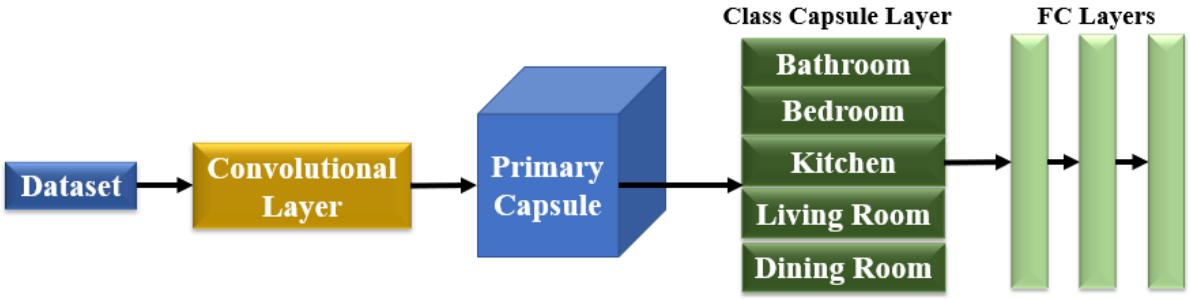


Figure 4.3. Capsule Neural Network (CapsNet) used for scene recognition

Table 4.1. Specifications for CapsNet

Layer	Specifications	Strides	Activation Function
Input	Image size: 128x128 (RGB scale)	-	ReLU
Convolutional Layer	Filters: 256 Channel Size: 9x9 Padding: Same	1	ReLU
Primary capsule	Number of capsules: 32 Number of channels: 64 Channel Size: 10x10 Padding: Valid	4	ReLU
Class Capsule	Number of capsules: 5 Number of channels: 16 Routings: 3	-	-
FC Layer -1	Number of Neurons: 512	-	ReLU
FC Layer-2	Number of Neurons: 1024	-	ReLU
FC Layer-3	Number of Neurons: 728	-	Sigmoid

The CapsNet (Chapter 3, section 3.4.1) is used first. The CapsNet has 6 layers. The convolutional layer is equipped with the ReLU activation function along with 256 filters, channel size of 9x9, and stride of 1. Keeping a stride of one increases the training time, but since the scene images have too many details, every pixel must be scanned. There are 32 capsules in the primary capsule layer used with 64 channels. The channels are used to distribute the capsules evenly. The channel size of the primary capsule is 10x10 with stride 4. There are 5-digit capsules in the digit capsule layer because there are only 5 classes in the indoor home scene dataset. There are 512 and 1024 neurons in the first and second FC layers, respectively and they both have a ReLU activation function. The last FC layer has 728 neurons with a sigmoid activation function. Figure 4.3 shows the discussed architecture of CapsNet used for implementing indoor home scene recognition and Table 4.1 shows the discussed parameters or specifications of CapsNet.

4.2.6. Results for traditional CapsNet

Table 4.2. The validation and testing accuracy for different deployed neural networks using 20,000 images (RGB) for training and 5000 images (RGB) for testing

Neural Network	Validation Accuracy (%)	Testing Accuracy (%)	Total Trainable Parameters	Testing Accuracy (%)	Training Time (hours)
CapsNet	**71**	**70**	**160 million+**	**92.4**	**38 (approx.)**
Faster RCNN	**76.9**	**74.2**	138 million+	**96.31**	28 (approx.)
Fast RCNN	75.3	73.6	140 million+	94.17	29 (approx.)
Mask RCNN	68.6	67.6	**60 million+**	93.22	**12 (approx.)**

After training the neural networks for scene recognition, different accuracies were produced. CapsNet produced a validation and testing accuracy of 71% and 70%, respectively. All the results are acquired on 20000 training images and 5000 testing images, which are evenly distributed in 5 different indoor home scenes. As per the obtained results shown in Table 4.2, it is clear that Faster RCNN produced the highest accuracy. CapsNet produced a comparable accuracy, which is the third-highest after Fast RCNN. Nonetheless, CapsNet has the highest total trainable parameters because the CapsNet appears to retain all information and does not just focus only on important features. Further, Mask RCNN took the least time to train and CapsNet took the highest time to train. VGG-16, VGG-19 and ResNet-152 were used as the CNN in Faster RCNN, Fast RCNN and Mask RCNN respectively. Moreover, the bounding box in Fast RCNN and Faster RCNN was not used, whereas, in Mask RCNN both bounding box and mask were not used because the task was not associated with object detection or recognition. Number of epochs used to train the mentioned neural networks is 100 with each

epoch having 100 iterations. In case of CapsNet, Faster RCNN, Fast RCNN and Mask RCNN, the mentioned accuracy was produced in 79th, 82nd, 75th and 77th epoch and after that the accuracy did not improve.

Table 4.3. The validation and testing accuracy for different deployed neural networks using 5000 images (RGB) for training and 1250 images (RGB) for testing

Neural Network	Validation Accuracy (%)	Testing Accuracy (%)	Testing Accuracy (%)
CapsNet	**71**	**70**	**91.99**
Faster RCNN	**69.1**	**68.2**	**95.43**
Fast RCNN	67	66	95.32
Mask RCNN	66.2	64.4	94.23

To analyse the performance of CapsNet on smaller datasets compared to other neural networks, the CapsNet and other neural networks were trained on just 5000 images and tested on 1250 images. It must be noted that all the images, in this case, are distributed evenly among the 5 classes of indoor home scenes. CapsNet produced the same accuracy of 71%. In contrast, the accuracy of Faster RCNN, Fast RCNN, and Mask RCNN dropped to 69.1%, 67%, and 66.2%, respectively. The results are shown in Table 4.3. This indicates that CapsNet does not require large datasets to train to a reasonable standard. This is important in cases where large data sets are not readily available. The performance of CapsNet drastically dropped when the number of images was reduced to 1500. The reason behind this reduction is the lack of presence of minimum required information for information extraction.

CapsNet produced comparable results despite having less deep architecture than the CNN-based architecture which has many layers and special techniques like batch normalization. The lower accuracy produced by CapsNet is because of the higher number of total parameters resulting in gradient explosion or vanishing and dying ReLU. The same does not happen in Faster RCNN because it has a relatively smaller total number of trainable parameters.

The total number of trainable parameters of CapsNet was found to be too high, 160 million+ (approx.). Therefore, training the CapsNet becomes difficult. The reduction in trainable parameters became therefore necessary. The images' dimension was changed from 256x256 (RGB) to 128x128 (grayscale) so as to reduce the trainable parameters. But when CapsNet was trained and tested on the changed image dimension, the accuracy of CapsNet also reduced. This led to the development of a new CapsNet structure called NoSquashCapsNet.

For a detailed illustration of how the CapsNet recognizes the indoor home scenes, we must refer to Figure 4.4. The bedroom scene shown in Figure 4.4(a) is the one that was not used in the training set. The CapsNet correctly recognized it. The kitchen scene, shown in Figure 4.4(b), was used for the training set. The CapsNet was unable to recognize this. In this case, a

threshold value of 65% was set and imposed using ROC (Receiver Operating Characteristic) [160]. This helps to indicate the CapsNet's minimum value of classification confidence. Any value of classification with less than 65% shows the message "don't know." This is important so that there is no confusion created by IAS that may directly or indirectly negatively impact the person getting assisted by it. The threshold value 65% indicates that a neural network is expected to produce an accuracy of at least 65%. In other words, this threshold value helps to reduce the number of false positive outputs.

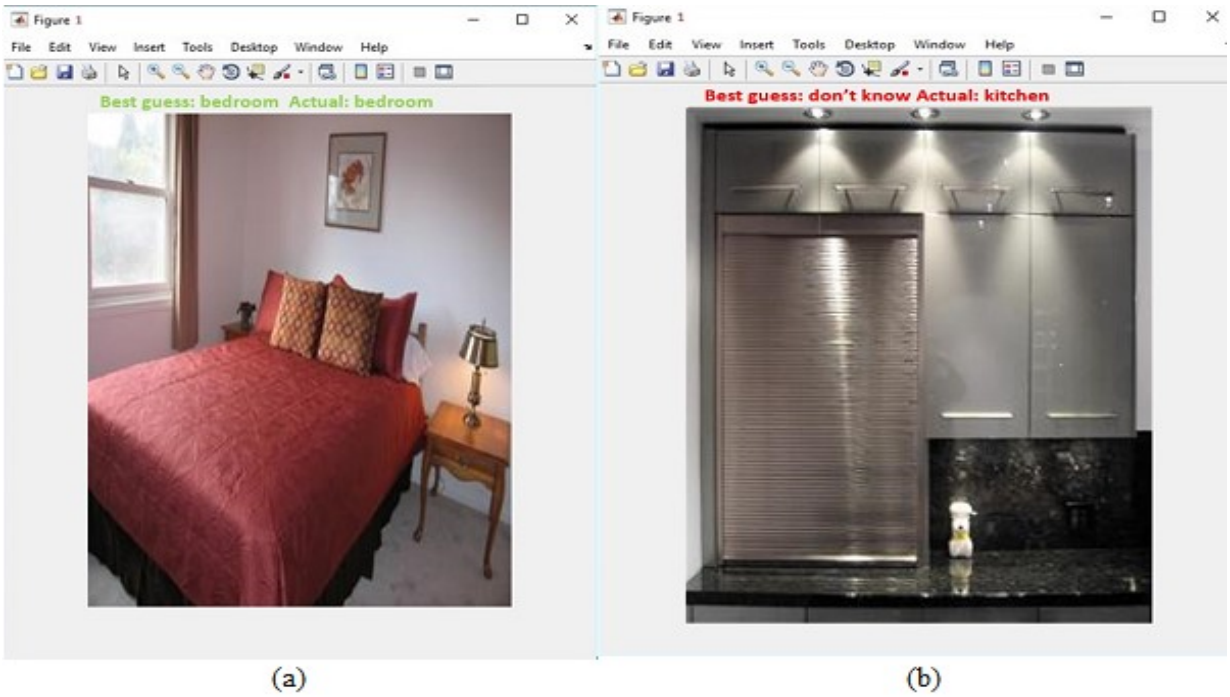


Figure 4.4. Images tested for image classification. (a) The bedroom scene (image not used in the training set) was tested in which the trained neural network was able to predict it correctly (b) A kitchen scene (image used in the training set), which the neural network was unable to recognize.

Table 4.4. Confusion matrix for traditional CapsNet

		Predicted				
	Scenes	**Bathroom**	**Bedroom**	**Dining Room**	**Living Room**	**Kitchen**
Actual	**Bathroom**	**87.1%**	2.3%	0.7%	0.75%	9.15%
	Bedroom	0%	**86.3%**	1%	10.6%	2.1%
	Dining Room	6.2%	15.2%	**39%**	31.5%	8.1%
	Living Room	1.8%	7.8%	28.9%	**53.4%**	8.1%
	Kitchen	2.8%	0.1%	10.6%	2.3%	**84.2%**

A confusion matrix obtained for CapsNet performance of indoor home scene is shown in Table 4.4. The green cells show the true-positive results and the red cells show the false-positive results. CapsNet produces best results when predicting the bathroom scene, in this case in 87.1% cases this was identified correctly. CapsNet has an accuracy of 86.3% and 84.2% for the bedroom and the kitchen scenes respectively. However, it does not perform well in the cases of the dining room and the living room. CapsNet accurately predicts only 39% of dining room scenes and it confuses the dining room with the living room in 31.5% of the cases, whereas the living room is confused with the dining room in 28.9% of the cases. This shows that the high similarity between scenes becomes a hinderance for CapsNet to learn subtle scene differences. It is also interesting to note that the kitchen is confused with the dining room in 10.6% of the kitchen images. This also shows that the presence of objects which can belong to both areas confuse the CapsNet. So, the presence of plates, spoons, bowls, etc., in the kitchen, which could also belong to a dining room presents difficulties for the network. In this case, CapsNet has not shown a good performance in learning the main objects associated with dining room scenes. CapsNet successfully differentiated between bathroom and bedroom scenes with only 2.3% of the bathroom scenes confused for bedroom scenes.

Table 4.5. Confusion Matrix for Faster RCNN

		Predicted				
	Scenes	**Bathroom**	**Bedroom**	**Dining Room**	**Living Room**	**Kitchen**
Actual	**Bathroom**	**96.2%**	3.37%	0.33%	0.1%	0%
	Bedroom	0%	**94.7%**	0.43%	4.87%	0%
	Dining Room	0.48%	0.93%	**20.67%**	15.94%	61.98%
	Living Room	0%	23.59%	6.47%	**66.53%**	3.41%
	Kitchen	0%	0%	6.92%	0.18%	**92.9%**

The confusion matrix of Faster RCNN, Fast RCNN and Mask RCNN are shown in Table 4.5, 4.6 and 4.7 respectively. When compared to traditional CapsNet all three have predicted the scenes much better. Faster RCNN, Fast RCNN and Mask RCNN predicted bathroom,

bedroom and kitchen scenes more accurately. However, traditional CapsNet has the best accuracy for predicting the dining room and the second-best accuracy, after Faster RCNN, for predicting living room scenes. From the confusion matrices it is also clear that all four architectures have failed in properly predicting with reasonable accuracy the dining room and living room scenes. This indicates a general lack of overall good performance in indoor environments which can be because all networks have confused objects in one scene with objects in the other, since the objects in both dining room and living room are closely related, e.g., tables and chairs.

Table 4.6. Confusion Matrix for Fast RCNN

		Predicted				
	Scenes	**Bathroom**	**Bedroom**	**Dining Room**	**Living Room**	**Kitchen**
Actual	**Bathroom**	**97.57%**	2.19%	0.04%	0.2%	0%
	Bedroom	0%	**91.79%**	0.21%	7.88%	0.12%
	Dining Room	0.2%	0.55%	**35.04%**	12.09%	52.12%
	Living Room	0.83%	40.33%	4.07%	**50.95%**	3.82%
	Kitchen	0%	0.04%	6.31%	1%	**92.65%**

Table 4.7. Confusion Matrix for Mask RCNN

		Predicted				
	Scenes	**Bathroom**	**Bedroom**	**Dining Room**	**Living Room**	**Kitchen**
Actual	**Bathroom**	**95.4%**	2.78%	0.42%	0.52%	0.88%
	Bedroom	0%	**96.85%**	0.55%	2.55%	0.05%
	Dining Room	0%	19.82%	**20.92%**	52.4%	6.86%
	Living Room	9.11%	31.36%	25.89%	**31.62%**	2.02%
	Kitchen	0%	0.51%	5.99%	0.29%	**93.21%**

4.3. NoSquashCapsNet: A Modified CapsNet

In the indoor home scene recognition using the CapsNet task, it is observed that the CapsNet has a higher number of total trainable parameters of around 160 million. This is too high and unacceptable because of the added training time taken by CapsNet when compared to CNNs. The indoor home scene images with dimensions 128x128 in the RGB scale were converted to grayscale images to reduce the total number of trainable parameters. However, when CapsNet was trained on grayscale images, the accuracy dropped to an unacceptable level. Higher trainable parameters also require much higher resources, which are not always readily available. Therefore, it became necessary to develop a new CapsNet architecture with fewer trainable parameters.

A new CapsNet architecture is developed and named NoSquashCapsNet. NoSquashCapsNet is an architecture formed using the concepts of CNNs and CapsNet. The main components of NoSquashCapsNet are the convolutional layers, Max Pooling layer, and a capsule layer without a squash function. The performance of NoSquashCapsNet is compared with CapsNet and CapsNet+ (the architecture of CapsNet+ is similar to NoSquashCapsNet but, in CapsNet+, the capsule retains the squash function). In training and testing the NoSquashCapsNet, the dataset is kept the same as that of the CapsNet.

The other main difference between NoSquashCapsNet and traditional CapsNet is that NoSquashCapsNet is deeper than traditional CapsNet. In traditional CapsNet, there is only one convolutional layer with ReLU activation for extracting all the required features from data whereas in NoSquashCapsNet there are multiple convolutional layers. This is to extract the maximum information from data so that the capsules do not find it difficult to learn the orientation of extracted features. Hence in NoSquashCapsNet, Max Pool layers are used to select the most prominent features of the data.

Figure 4.5. The conversion of 256x256 (RGB) image to 128x128 (Grayscale) image

For implementing the work, the same dataset of 20000 training images and 5000 testing images taken from the Places365 dataset [105] are used. The images are evenly distributed over

the five different scenes of indoor home. The scene categories are also the same, bedroom, bathroom, living room, kitchen, and dining room. The RGB images with dimension 256x256 are converted to RGB images with dimension 128x128 using shrinking process [159]. Then the RGB images with dimension 128x128 are converted to grayscale images of the same dimension. The RGB to Grayscale conversion is performed using weighted method or luminosity method (Grayscale image = 0.21R + 0.72G + 0.07B, where, R, G and B are red, green and blue respectively) [159]. All this is done to reduce the total number of trainable parameters. The complete process of conversion of images is shown in figure 4.5. The converted images are used for training and testing the developed NoSquashCapsNet architecture.

4.3.1. Proposed NoSquashCapsNet Architecture

As discussed, the capsule i.e., the main unit for CapsNet, has three basic functions: affine transformation, the sum of weights, and squash function. The affine transformation and sum of weights are essential functions of capsules. These two functions serve the primary purpose of capturing the orientation of images. The squash function is responsible for performing the unit scaling of images. The squash function reduces the information vector to between 0 and 1 but it retains all the information and therefore the CapsNet architecture is cumbersome to train with too many parameters. CapsNet does not appear to have the means to allow the learning parameters to concentrate on fewer but the most prominent features of the presented information.

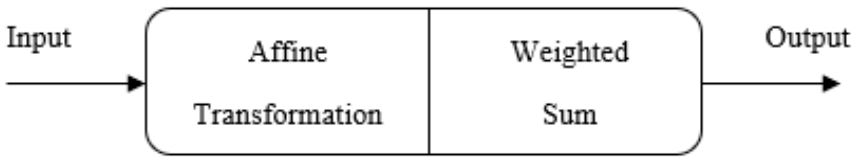


Figure 4.6. A modified capsule without squash function

Moreover, the squash function keeps the direction of the vectors undisturbed which appears not to be the best possible way for learning indoor home scenes. Therefore, the squash function was removed. Introducing Max Pool layers in CapsNet helps in acquiring the important information and discarding information which is not significant, thus making the training faster and less complicated. Hence, the squash function was removed and the Max-Pool function was introduced. The architecture of the capsule without squash is shown in Figure 4.6. The combination of using both the Max-Pool layer and the squash function is also tested (this is the CapsNet+ architecture).

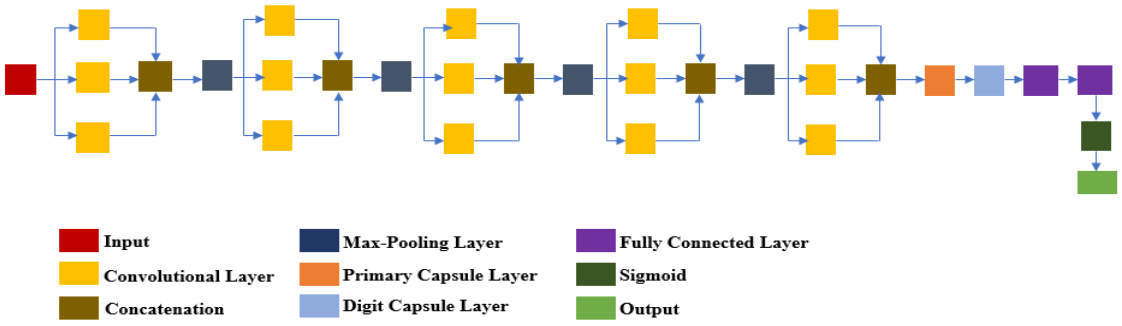


Figure 4.7. Modified Capsule Neural Network (NoSquashCapsNet)

The NoSquashCapsNet architecture has 14 layers. There are five layers (layers one, three, five, seven, and nine of NoSquashCapsNet) of three concatenated convolutional layers. Each concatenated convolutional layer is connected to a subsequent Max Pooling layer (layers two, four, six, and eight of NoSquashCapsNet). Only the concatenated convolutional layer in the ninth layer of NoSquashCapsNet is not connected to a Max Pool layer. The ninth layer, the concatenated convolutional layer, is directly connected to the primary capsule layer. The eleventh layer of NoSquashCapsNet is the digit capsule. The twelfth and thirteenth layers are the FC layers. The last layer of NoSquashCapsNet is the classifier layer. The classifier here is a sigmoid function. The explained architecture is shown in Figure 4.7.

Table 4.8 provides information on different hyperparameters of different layers of NoSquashCapsNet architecture. Cov1_1, Conv1_2, Conv1_3…...Conv5_1, Conv5_2, Con5_3 are concatenated convolutional layers in the first, third, fifth, seventh and ninth layers of NoSquashCapsNet, respectively. The reason behind having three convolutional layers in each layer with same dimension, is to establish feature extraction from whole data parallelly (Each convolution extracts feature from whole data in every layer). This helps to extract the features properly from large and complex data. The clearer the features are, the better is the learning capacity of any neural network.

Keeping a stride of 1 helps in the proper acquisition of information by every convolutional layer. The padding of each convolutional layer is kept unchanged. The motivation behind keeping the padding unchanged is to keep the dimension of input and output equal. All the Max Pooling layers have 2x2 channel size with padding unchanged and stride of 2. Max Pool is here for the down-sampling of input representation. Both convolutional layers and Max Pooling layers have a smaller channel size to capture complex and smaller features from the data. This is because of the higher amount of minute information present in the indoor home scene recognition. Max pool layer also helps in extracting only the relevant information. The irrelevant information is discarded. This prevents the neural network from getting

overwhelmed with information and helps the capsule to extract the relationship between the extracted features without any confusion.

Table 4.8. NoSquashCapsNet specification table

Layer	**Name**	**Specifications**	**Strides**
Input		Image size: 128x128 (Grayscale)	-
Layer-1	Conv 1_1 Conv 1_2 Conv 1_3	Output Concatenated Filters: 64 Channel Size: 3x3 Padding: Same	1
Layer-2	Max-Pool	Channel Size: 2x2 Padding: Same	2
Layer-3	Conv 2_1 Conv 2_2 Conv 2_3	Output Concatenated Filters: 128 Channel Size: 3x3 Padding: Same	1
Layer-4	Max-Pool	Channel Size: 2x2 Padding: Same	2
Layer-5	Conv 3_1 Conv 3_2 Conv 3_3	Output Concatenated Filters:256 Channel Size: 3x3 Padding: Same	1
Layer-6	Max-Pool	Channel Size: 2x2 Padding: Same	2
Layer-7	Conv 4_1 Conv 4_2 Conv 4_3	Output Concatenated Filters: 512 Channel Size: 3x3 Padding: Same	1
Layer-8	Max-Pool	Channel Size: 2x2 Padding: Same	2
Layer-9	Conv 5_1 Conv 5_2 Conv 5_3	Output Concatenated Filters: 512 Channel Size: 3x3 Padding: Same	1
Layer-10	Primary capsule	Number of capsules: 32 Number of channels: 16 Padding: Valid	2
Layer-11	Digit Capsule	Number of capsules: 5 Number of channels: 16 Routings: 3	-
Layer-12	FC Layer -1	Number of Neurons: 4096	-
Layer-13	FC Layer-2	Number of Neurons: 4096	-
Layer-14	FC Layer-3	Number of Neurons: 5	
Layer-15	Sigmoid Function		-

The primary capsule is on the tenth layer of NoSquashCapsNet. In the primary capsule layer, there are 5 capsules with 16 channels. The stride in the primary capsule is 2, with padding being valid. The eleventh layer is the digit capsule layer. The number of primary capsules is also decided based on the complexity of information. A higher level of complexity in information requires more capsules. However, it must be noted that a higher number of capsules increase the total number of training parameters that directly impacts the training/learning by making it slow and also by decreasing the efficiency of the neural network.

There are 5 digital capsules because there are 5 different indoor home scenes with channel size 16. The number of iterations of routing is 3. The twelfth, thirteenth, and fourteenth layers are the fully connected layers. FC layers in the twelfth and thirteenth layers have 4096 neurons, and the FC layer in the fourteenth layer has 5 neurons. The feature space helps in deciding the number of neurons in the FC layers. However, the exact number of neurons depends mostly on thorough experimental analysis to help neural networks work efficiently. The last layer, i.e., the fifteenth layer, is the sigmoid function, which is the classifier.

4.3.2. Results for NoSquashCapsNet

As per the analysis, it is clear that NoSquashCapsNet performed better and more efficiently in learning and recognising indoor home scenes. The accuracy of CapsNet on grayscale images shows that it ultimately failed to extract any information on the dataset. The accuracy of NoSquashCapsNet increased significantly as compared to CapsNet performance on grayscale images, shown in Table 4.9. Number of epochs used to train the mentioned neural networks is 100 with each epoch having 100 iterations. For CapsNet+ and NoSquashCapsNet, the mentioned accuracy was produced in 81st and 85th epoch respectively and after that the accuracy did not improve.

Table 4.9. Validation and Testing accuracy of CapsNets

Neural Network	Validation Accuracy (%)	Testing Accuracy (%)	Trainable Parameters (approx.)	Testing Accuracy (%)	Testing Time (hours)
CapsNet on grayscale images	20	17.2	53 million+	72.11	10 (approx..)
CapsNet on RGB scale images	71	70	160 million+	92.4	36 (approx..)
CapsNet+ on grayscale images	64.7	64	30 million+	93.1	4 (approx.)
NoSquashCapsNet on grayscale images	**70.8**	**70**	**30 million+**	**94.18**	**4 (approx.)**

This shows that the Max Pool helps in the acquisition of significant relevant information. Therefore, it can be concluded that the information processed by convolutional layers and Max-Pooling layers help the affine transformation and weighted sum to work better. Removing the squash function restricts the capsule from doing the unit scaling and keeping the direction of information unchanged. The NoSquashCapsNet and the CapsNet+ were also trained and tested on RGB images of dimension 256x256 and 128x128. However, the total number of trainable parameters remained very high (50 million+ approx.) and the accuracy produced in both these cases remained low (50% approx. in both cases).

According to Table 4.9, CapsNet registered validation and testing accuracy of 20% and 17.2%, respectively, on grayscale indoor home scene images with dimension 128x128. The total trainable parameters, in this case, were reduced to 53 million+ (approx.). The same CapsNet produced validation and testing accuracy of 71% and 70%, respectively, on RGB scale indoor home scene images with dimension 256x256. However, its efficiency remained low because of the higher total number of trainable parameters, i.e., 160 million+ (approx.).

When used with CapsNet+, the grayscale indoor home scene images produced validation and testing accuracy of 64.7% and 64%, respectively. The validation and testing accuracy of NoSquashCapsNet is 70.8% and 70%, respectively. In both cases, the total number of trainable parameters was around 30 million+ (approx.). Therefore, the difference between the validation accuracies of CapsNet on RGB scale images and NoSquashCapsNet on grayscale images is just 0.2%, where the testing accuracy remains the same. However, the efficiency of NoSquashCapsNet is much higher than the CapsNet's efficiency (trained and tested on RGB scale images). This is because NoSquashCapsNet produces the same accuracy with fewer trainable parameters.

The confusion matrix for NoSquashCapsNet that shows the accuracy for each class is shown in Table 4.10. NoSquashCapsNet is able to predict the maximum number of bedroom scenes as it has the highest true-positive percentage of 85.9%. Like traditional CapsNet, NoSquashCapsNet also does not confuse any bedroom scene with bathroom. In addition to this, NoSquashCapsNet also does not confuse any living room scene with bathroom. However, NoSquashCapsNet confused 5.25% of bathroom scenes with bedroom and 3.05% of bathroom scenes with living room. NoSquashCapsNet confuses the living room scenes with dining room as 47.2% of living room scenes were wrongly predicted as dining room. Furthermore, NoSquashCapsNet confuses dining room with living room and kitchen. NoSquashCapsNet also confuses 10.1% of Bedroom and 15.2 % of kitchen scenes are confused with dining room.

However, the important thing to note from the confusion matrix presented in Table 4.10 is that true-positive outputs for both the dining room and living room increased as compared to CapsNet. There is a dip in true-positive percentage in bathroom, bedroom and kitchen but is very minimal.

Table 4.10. Confusion matrix for NoSquashCapsNet

		Predicted				
	Scenes	**Bathroom**	**Bedroom**	**Dining Room**	**Living Room**	**Kitchen**
Actual	**Bathroom**	**80.1%**	5.25%	1.7%	3.05%	9.9%
	Bedroom	0%	**85.9%**	10.1%	0.2%	3.8%
	Dining Room	0.15%	0.85%	**50.1%**	26.6%	22.3%
	Living Room	0%	0.5%	47.2%	**52.3%**	0%
	Kitchen	0.35%	0.75%	15.2%	2.1%	**81.6%**

4.4. Conclusion

Overall, in this chapter, a traditional CapsNet architecture is implemented for indoor home scene recognition. Further, to improving CapsNet, especially in terms of reducing the total number of parameters, a new CapsNet called NoSquashCapsNet was developed. During testing it was found that, even though the overall accuracy of the NoSquashCapsNet remained below other network structures, this was still a comparable performance and the proposed neural network does have the advantages of an overall simpler architecture, fewer parameters and requiring less training data to reach comparable overall accuracy.

In the next chapter, the CapsNet architecture is applied to performing object recognition. Armed with the already acquired knowledge on CapsNet and NoSquashCapsNet performances, improvements are made using two new CapsNets architectures, namely 1D-CapsNetA and 1D-CapsNetB which are developed and tested on 3D object datasets.

Chapter - 5

1D CapsNets for Efficient 3D Indoor Home Object Recognition

5.1. Introduction

In this chapter, CapsNets are used to single 3D object recognition since these types of networks are good in learning the orientation of 3D objects – a vital process for indoor IAS applications. For object detection and recognition, the methods used for most neural networks involve neural network architectures trained on 2D image datasets. This process, however, appears to have two main limitations. The first limitation is `that there is no guarantee that a neural network will recognize an object in an orientation other than the orientation which was presented to the network during training. The second limitation arises because of the first limitation. To produce higher accuracy, very large datasets of 2D images are required of the same object in the same condition but in different orientations must exist. However, large datasets with 2D images with multi-views are not always possible to obtain or easy to handle. CapsNets require less data to learn about objects because they do not require images with multiple views of objects. Therefore, efficient training using smaller datasets becomes possible.

Labelling objects in an image using annotations is another problem associated with datasets. This problem leads to confusion in the neural network during training because of background inclusion in the annotated objects. Background overlap can happen when multiple objects are labelled using annotation in the same image. This background overlap which can appear under different labels can confuse a neural network during training. So, when a system looks at an object to recognize it from different angles then, anything else falling in the background of the object creates a problem in proper recognition. For better efficiency a requirement to have neural networks which can extract feature information and proper orientation from 3D image datasets are proposed in different research. In the 3D representation of objects, each image consists of nothing else except one object. This helps the system to understand the object by looking at it from any view point without having the background issues mentioned above.

In this chapter, CapsNets are developed that are capable of efficiently recognizing 3D indoor home objects from 3D indoor images, so as to assist indoor robotic devices in performing home

tasks. Three CapsNet architectures involved are, PointCapsNet, 1D CapsNetA and 1D CapsNetB. In this chapter therefore, section 5.2 is about PointCapsNet. Section 5.3 introduces the developed 1D CapsNet A and 1D CapsNetB architectures for 3D indoor home object recognition. Section 5.4 discusses all the results produced by the developed techniques and section 5.5 concludes the chapter.

5.2. Proposed PointCapsNet Architecture

PointCapsNet is inspired by the PointNet architecture. The PointNet architecture is discussed in Chapter-3, section 3.3.6. The original design was developed to recognise 3D objects after getting trained on point cloud datasets in a 1D array [120]. Point cloud dataset can be described as a raw 3D dataset that represents any object using a collection of points. Visual representation of point cloud data of a torus object is shown in Figure 5.1, where each point is used to represent the surface of the object.

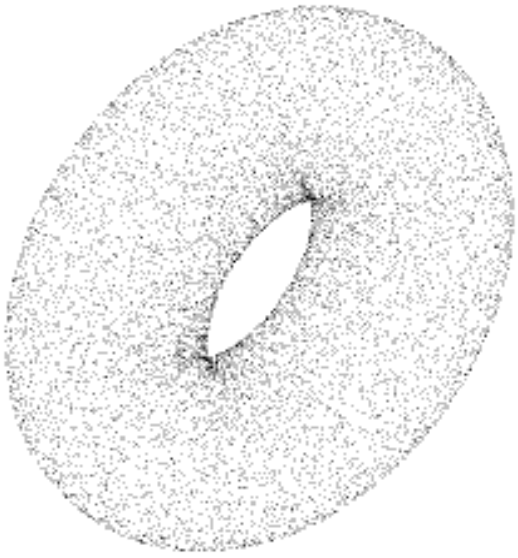


Figure 5.1. An object presented in point cloud format

The point cloud form of the dataset needs to be converted to 3D voxel grids in most cases in order to train any neural network on 3D datasets — this conversion however results in increased trainable parameters. A 3D voxel grid is the representation of objects in 3D so that its x, y, and z coordinates could be known. It is also termed as a counterpart of pixels in 2D images. Figure 5.2 (a) is the point cloud representation of a toy. The point cloud data is converted into 3D voxel grid format, shown in Figure 5.2 (b). It can be clearly seen that there are different voxels in form of cubes that are used to represent the toy. These cubes have x, y and z coordinates.

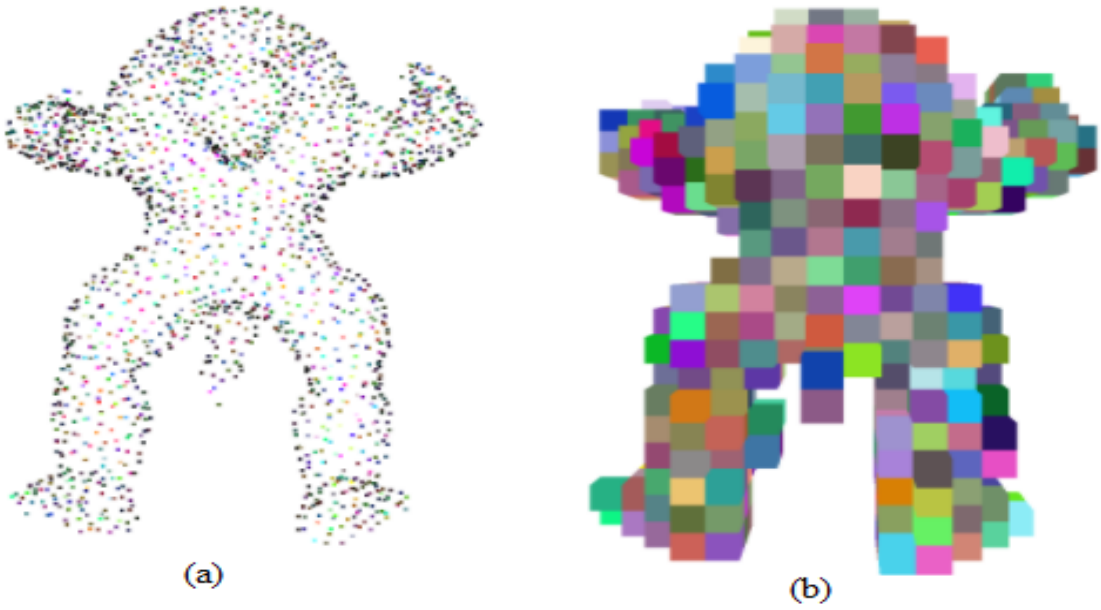


Figure 5.2. (a) An animal shown in point cloud data and (b) The same animal shown in voxel grids

Therefore, the reasons behind using point cloud data are to have lower trainable parameters and to make the learning process of neural network faster without affecting the overall accuracy. Point cloud data formats are also used in metrology, modelling of manufacturing parts, quality inspection, mass customisation and animation. because point cloud data are supposed to carry the very minimum information of any image which makes the task of analysing and implementing different tasks easier.

ModelNet-10 and ModelNet-40 datasets are used in this research [117], and both the datasets are in point cloud form. ModelNet-10 and ModelNet-40 datasets are first converted from 3D datasets to 1D array datasets. ModelNet-10 is a subset of ModelNet-40 and has all ten categories of indoor home objects. ModelNet-40 has forty categories of 3D objects, but these categories represent other objects too - not just indoor home objects, like car, airplane and tent. ModelNet-40 is not directly related to this work but it is being used here so that comparisons with other works can easily be made.

In case of PointNet, the point cloud data conversion is not required. Therefore, PointNet became the base of further development. However, the PointNet architecture is very complicated because the complete architecture is combination of different neural networks. The input transformation and feature transformation in Figure 5.4 are performed using T-Net which is shown in Figure 5.3. T-Net is a regression network [161] which is implemented in PointNet to predict 3x3 transform matrix [120]. This further helps to predict the affine transformation matrix at input and feature transform layers by applying coordinates of the input points. Only predicting affine transformation is not enough to acquire the orientation

information of 3D objects and also it does not help in acquiring information of an object from different viewpoints. Further, MLP which itself is a feedforward ANN is also used in different layers of the PointNet architecture (Chapter-3, section 3.3.6, Figure 3.19). The overall accuracy produced by PointNet is also low (77.6% & 89.2% on ModelNet-10 and ModelNet-40 datasets respectively).

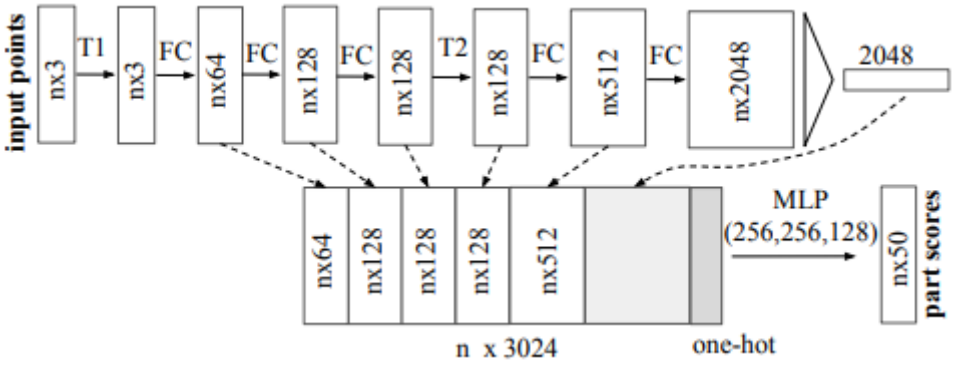


Figure 5.3. T-Net Architecture [120]

This led to the development of PointCapsNet shown in Figure 5.4, which is a modified PointNet architecture enhanced with 1D capsule (primary capsule and digit capsule discussed in Chapter 4, section 4.2.1), so that it can be directly trained on point cloud data. The conversion to a 1D array of 3D data (in point cloud form) happens before the data enters the first layer of neural network for the training process. Therefore, the number of specified points required to be trained at once for all architectures presented in this chapter is 1024, which is represented as n in Figure 5.3 and 5.4. This helps in substantially decreasing the trainable parameters because there is no need to specify the data's input dimension. MLP block in Figure 5.4 is for identically mapping the higher-dimensional space to input points [120].

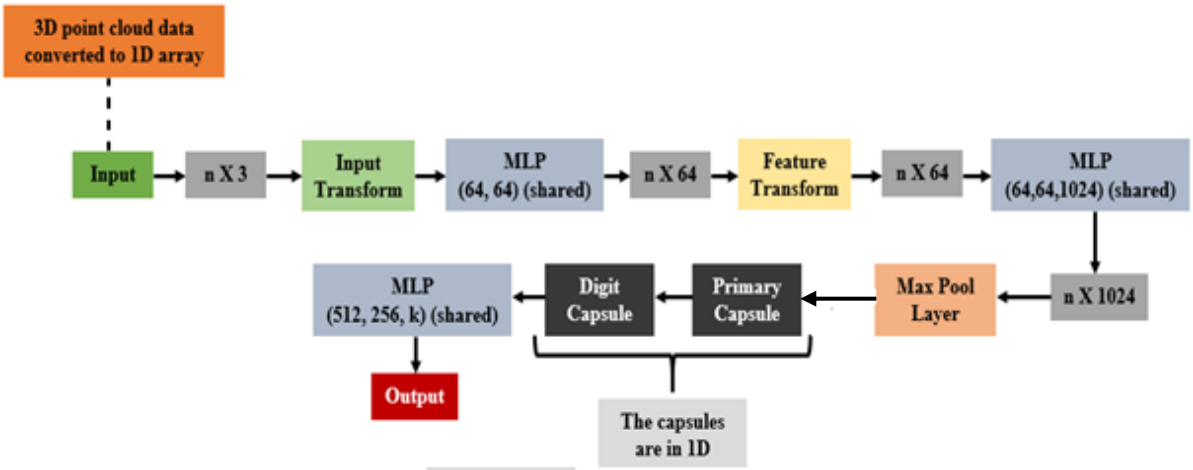


Figure 5.4. PointCapsNet (Point Capsule Neural Network) Architecture

Max-Pool layer in Figure 5.4 is used to extract the most relevant features and encode the global feature vectors [120]. Furthermore, the integration of the 1D capsules is done to properly acquire the orientation information of 3D objects from different viewpoints. During testing, the performance of PointCapsNet showed significant improvement over the PointNet architecture for the relevant ModelNet-10 dataset but also a drop in accuracy for the ModelNet-40 (89.43% and 83.4% respectively). Despite this improvement and change in architecture, PointCapsNet retained some of the problems that existed in PointNet like capturing of orientation of objects using T-Nets and presence of combination of different neural networks which did not help with either training or further performance improvement. Therefore, this led to the development of different CapsNet architectures that have simpler architecture and improved performance.

5.3. Proposed 1D CapsNet Architectures: 1D CapsNetA and 1D CapsNetB

Despite the improvement shown by the PointCapsNet architecture, it was felt that by modifying the CapsNet architecture outlined in Chapter 4, a simpler architecture with better accuracy and less trainable parameters could be achieved. The decision was also taken because of the limitations or issues associated with PointCapsNet architecture, outlined in the previous section.

The CapsNet architecture with Max-Pool layer was introduced in Chapter 4 which indicated superior efficiency in indoor scene recognition. It was therefore decided to use the same approach to accomplish an efficiency improvement using CapsNet for indoor object recognition. Furthermore, it was decided to also use some of the methods utilised in the PointCapsNet architecture which helped to significantly improve performance in indoor object datasets as shown in the previous section.

A large number of trainable parameters are inevitable in 3D neural networks. The conversion of the 3D data to 1D array helped to partly mitigate the problem of having many trainable parameters. Therefore, a similar conversion of 3D data in 1D array format is used in this case to train the neural networks on the 3D datasets. The CapsNet architectures are, therefore, also converted to 1D to use the information from 1D array datasets and process it accordingly.

Two CapsNet architectures, 1D CapsNetA, and 1D CapsNetB were developed, which are shown in Figure 5.5(a) and 5.5(b) respectively. The position of the Max-Pool layer used is different in these architectures, which is the only difference between the two 1D CapsNets. The conversion of CapsNet to accommodate the 1D data meant that a 1D convolutional layer, a 1D Max-Pool layer, and a 1D capsule combination are used for both the 1D CapsNet

architectures. With reference to Figure 5.5(a), which shows the schematic diagram of the 1D CapsNetA architecture, each of the first four layers of this architecture, comprises two convolutional layers which have concatenated outputs. The input to the subsequent two convolutional layers is the concatenated output from the previous convolutional layer. The number of filters in convolutional layers in the first, second, third and fourth layers are 64, 128, 256, and 512, respectively. The filter size of all convolutional layers is 3. The initial convolutional layers help in extracting the low-level features (like curves, edges and lines). In the later convolutional layers, the higher-level features (like semicircle, squares and circle) are extracted using the already extracted low-level features. The Max-Pool layer extracts the most prominent features and forwards the data to the capsules. The motivation behind placing the Max-Pool layer after all the concatenated convolutional layers and just before the capsule in this architecture, is to extract the prominent information only after the higher-level features are obtained and thus test the accuracy of this architecture in this scenario.

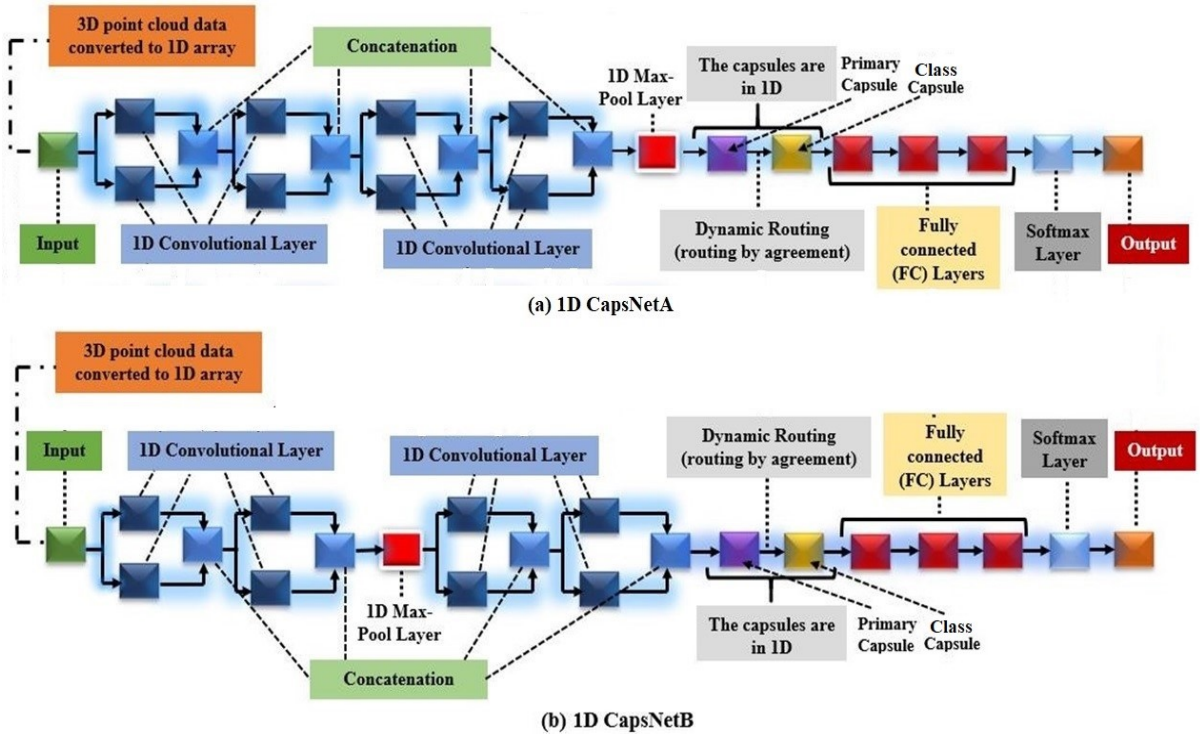


Figure 5.5. Architectures of One-dimensional Capsule Neural Networks (1D CapsNets) with Max-Pool (a) 1D CapsNetA, (b) 1D CapsNetB

The primary capsule layer is responsible for extracting the orientation of the features and then resize them. There are 32 capsules in the primary capsule layer, and the channel size is 16 (the working of capsules is discussed in Chapter 4, section 4.2.3 and section 4.2.4). The primary capsules are equally distributed over the channels. In other words, each channel has two

primary capsules. The overall arrangement was achieved empirically after trying and testing several different structures. Before finalising 1D CapsNetA and 1D CapsNetB, the different arrangements implemented included the integration of Max-Pool layer after every concatenated convolutional layer, integration of Max-Pool layer after layer 1 and layer 3 and use of three concatenated convolutional layers in every layer instead of 2. However, the accuracy remained between 30% to 40%. The acquired information by the primary capsule layer is then sent to digit capsules using the dynamic routing process (Chapter 4, Section 4.2.4).

The information received by the digit capsules is stored as per the class. The same can be understood from Figure 5.6, which shows that the lower-level features present in lower-level capsules (primary capsule layer) are transferred to the higher-level capsules (Digit Capsule layer) which stores higher-level features using routing by agreement. The number of cells in DigitCaps is 10 and 40 for ModelNet-10 and ModelNet-40, respectively. This is because ModeNet-10 has ten classes, and ModelNet-40 has forty classes. The channel size of DigitCaps is also 16.

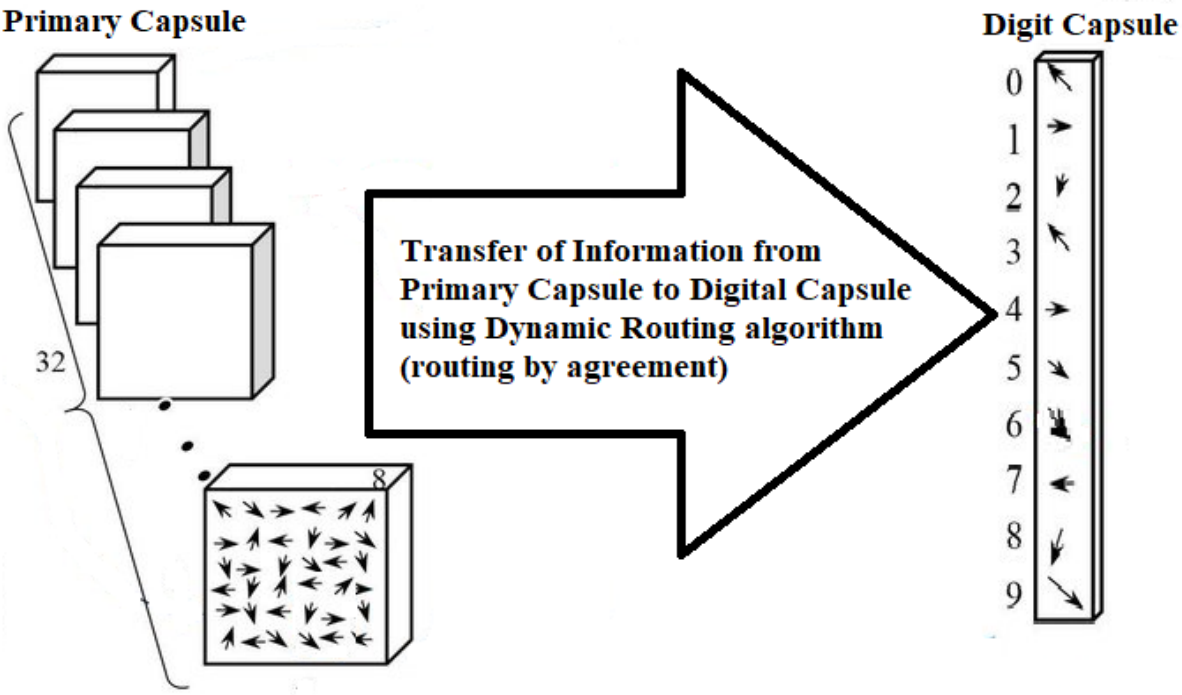


Figure 5.6. Transfer of information from Primary Capsule to Digit Capsule

Subsequently, the Fully Connected (FC) layers receive all the information from DigitCaps. There are 4096 neurons in the first and second FC layers. There are five neurons in the third FC layer. ReLU activation function is used in all convolutional layers and FC layers. For a proper output, the softmax layer is incorporated in the architecture so that the output of the neural networks is probability distribution.

Table 5.1. Complete specifications of 1D CapsNetA and 1D CapsNetB

Name	**Specifications**	**1D CapsNetA Layers**	**1D CapsNetB Layers**	**Strides**
Input	1D ModelNet-10 and ModelNet-40 dataset	Yes	Yes	-
Conv 1_1 Conv 1_2	Output Concatenated Filters: 64 Channel Size: 3	Yes, Layer-1	Yes, Layer-1	1
Conv 2_1 Conv 2_2	Output Concatenated Filters: 128 Channel Size: 3	Yes, Layer-2	Yes, Layer-2	1
Max-Pool	Channel Size: 2 Padding: Same	No	Yes, Layer-3	1
Conv 3_1 Conv 3_2	Output Concatenated Filters: 256 Channel Size: 3	Yes, Layer-3	Yes, Layer-4	1
Conv 4_1 Conv 4_2	Output Concatenated Filters: 512 Channel Size: 3	Yes, Layer-4	Yes, Layer-5	1
Max-Pool	Channel Size: 2 Padding: Same	Yes, Layer-5	No	1
Primary capsule	Number of capsules: 32 Number of channels: 16 Channel Size: 3 Padding: Same	Yes, Layer-6	Yes, Layer-6	1
Digit Capsule	Number of capsules: 10&40 Number of channels: 16 Routings: 3	Yes, Layer-7	Yes, Layer-7	-
FC Layer -1	Number of Neurons: 4096	Yes, Layer-8	Yes, Layer-8	-
FC Layer-2	Number of Neurons: 4096	Yes, Layer-9	Yes, Layer-9	-
FC layer - 3	Number of Neurons: 5	Yes, Layer-10	Yes, Layer-10	-
Softmax Function	-	Yes, Layer-11	Yes, Layer-11	-

The work and testing carried out for the 1D CapsNetA architecture helps to form a hypothesis that low-level features are considerably more than higher-level features. Hence, if instead of propagating all the low-level features (many of which are duplicated in different

orientations) to the stages of the higher-level feature extraction, the prominent low-level features are acquired earlier in the process, which is then likely to make the extraction of higher-level features easier and far more efficient. On this basis therefore, the second architecture proposed is the 1D CapsNetB shown in Figure 5.4(b). In Figure 5.4(b) then, the Max-Pool layer is placed earlier in the overall structure and just after the concatenated convolutional layers of the second layer. In this case, the prominent features are extracted only after the first two sets of concatenated convolutional layers. So, the important low-level features are pooled at an earlier stage. The next two concatenated convolutional layers extract the higher-level features and send the extracted features to capsules. Here the higher-level features are less and therefore, the Max-Pool layer is not required after the 5^{th} layer and also because all the high-level features are needed to properly learn the orientation of each object.

In layers 1, 2, 3 and 4 of 1D CapsNetA, (Conv1_1 and Conv1_2), (Conv2_1 and Conv2_2), (Conv3_1 and Conv3_2) and (Conv4_1 and Conv4_2) are the concatenated convolutional layers respectively presented in Table 5.1. The same concatenated convolutional layers in 1D CapsNetB are in layers 1, 2, 4, and 5, respectively. The dimension of all convolutional layers, Max-Pool layer, and capsules are of 1D. 1D CapsNetA and 1D CapsNetB have the same technical specifications. Padding in technical term is kept ‘same’ so that in Max-Pool layer the input has dimension equal to the output dimension. This is done to utilize the information acquired from convolutional layers in their original form.

5.4. Results

The presented neural networks’ architectures have fewer trainable parameters than other available neural networks, i.e., each 1D CapsNet has just over 1 million trainable parameters to train. On both ModelNet-10 and ModelNet-40 datasets, the best accuracy was produced by 1D CapsNetB. This proves that applying the Max-Pool layer works better when applied earlier in the process. Notably, comparable outputs have also been produced by PointCapsNet. However, on the ModelNet-40 dataset, the accuracy PointCapsNet produced was rather low by comparison. The use of MLP in the initial layers of architecture instead of convolutional layers for feature extraction can be attributed to the reason behind the lower accuracy level, in this case.

The performance of many state-of-the-art architectures have been surpassed by the architectures presented in this chapter in terms of accuracy. However, there are some other architectures [124-128] that have slightly better accuracy than the proposed 1D CapsNets

architectures developed in this chapter. Comparing these architectures, it should be noted that better performance is achieved at considerable cost for a number of reasons.

These reasons include: a) a very high number of training parameters which is between 2 to 250 times the number of parameters present in the 1D CapsNets architectures; b) the requirement of converting point cloud data into 3D voxel grids, which is one of the main causes of increased number of trainable parameters; and c) using multiple 2D images (from different angles) for accurately describing 3D objects which also leads to increase in data and parameters. Thus, despite slightly higher accuracies, there are problems present on efficiency grounds for these architectures.

Table 5.2 shows different neural networks' benchmark accuracies on ModelNet-10 and ModelNet-40 datasets, the accuracy of the architectures presented in this chapter, the total number of parameters for each network, and whether multi-view images and point cloud to 3D voxel conversions are required. 1D CapsNetA, 1D CapsNetB and PointCapsNet produced accuracies of 92.03%, 91.48% and 89.43% respectively on ModelNet-10, whilst on ModelNet-40 the accuracies produced are 91.04%, 89.97% and 83.4% respectively. 1D CapsNetA and 1D CapsNetB are developed and trained directly on point cloud datasets, which can perform 3D object recognition tasks. It must be noted that 1D CapsNets architectures have been based on entirely different concepts in order to develop neural networks with good accuracies and far better efficiency.

From Table 5.2 it is seen that RotationNet (discussed in Chapter-3, section 3.3.6) displays the highest accuracy on ModelNet-40 and ModelNet-10 datasets. However, this network uses multiple views of 2D images to accurately represent 3D objects. The network thus assumes an availability of multiple view images for all 3D objects which is generally not true. The size of the dataset used by RotationNet is also much larger. This is contrary to the objective of achieving better accuracy on a smaller dataset given that, in most practical applications, large datasets are unlikely to be readily available. The training and learning speed of RotationNet are also affected because of the number of trainable parameters which is five times higher than the 1D CapsNets. From Table 5.2 it is also seen that different CapsNets (3D CapsNet Architectures 1 and 2, and 3D-CapsNets) can be trained only after the point cloud data is converted to 3D voxel grids. All the CapsNets have reported very high total number of trainable parameters when compared to the developed 1D CapsNets. In the case of Achlioptas et al. [123] (Chapter 3, section 3.3.6), the design uses GANs which are also directly trained on point

cloud data. However, because of the property of the GANs to generate fake images using random noise, the data gets increased (almost doubled) during the learning process.

Table 5.2. Comparison of different architectures using ModelNet-40 and ModelNet-10 datasets

Neural Network	**ModelNet-40 Accuracy (%)**	**ModelNet-10 Accuracy (%)**	**Total Number of Parameters**	**Multi-view images required**	**Point cloud to 3D voxel conversion required**
1D CapsNetB	**91.04**	**92.03**	**1 million (approx.)**	**No**	**No**
1D CapsNetA	**89.97**	**91.48**	**1 million (approx.)**	**No**	**No**
PointCapsNet	**83.4**	**89.43**	**1 million (approx.)**	**No**	**No**
3-D CapsNet Architecture-1 [131]	88.67	91.48	200 million (approx.)	No	Yes
3-D CapsNet Architecture-2 [131]	89.66	91.37	200 million (approx.)	No	Yes
3D Capsule [132]	92.7	94.7	150 million (approx.)	No	No
3D-CapsNets [130]	82.73	93.08	250 million (approx.)	No	Yes
RotationNet [116]	97.37	98.46	5 million (approx.)	Yes	No
Achlioptas et al. [123]	84.5	95.4	2 million (approx.)	GAN used (Data increases)	No
PointNet	89.2 [120]	77.6 [119]	880K (approx.)	No	No

The confusion matrices display the accuracy for the 1D CapsNetA and 1D CapsNetB on each class of objects in the ModelNet-10 in Tables 5.3 and 5.4 respectively. The green cells in both tables show the true positive results, i.e., the correct prediction percentage. The red cells in both tables show the false-positive results, i.e., the wrong prediction percentage. The true

positive values of confusion matrices show that most of the predictions for each class were correct. However, the false-positive results for each class obtained show that some closely related objects were wrongly detected. For example, in Table 5.3, the object table, was confused with the object desk in 9.47% of cases.

Table 5.3. Confusion matrix for 1D CapsNetA on ModelNet-10 (values are in %)

	Predicted									
	Objects	Bathtub	Bed	Chair	Desk	Dresser	Monitor	Night Stand	Sofa	Table	Toilet
Actual	Bathtub	92.9	0	3.2	0	0	0	0	0	0	3.9
	Bed	0	90.2	0	0	0.3	0	0	5.56	4.12	0
	Chair	0	0	93.7	0	o	0	0	0	0	6.3
	Desk	0	0	0	88.4	0	0	0	0	11.6	0
	Dresser	0	0	0	11.24	85.6	0	0	0	3.16	0
	Monitor	0	0	2.23	3.27	0	89.3	0	0	5.2	0
	Night Stand	0	0	0	14.72	0	0	84.1	0	4.18	0
	Sofa	0	0.81	8.24	0	0	0	0	90.95	0	0
	Table	0	0	0	9.47	0	0	0	0	90.53	0
	Toilet	5.96	0	0.02	0	0	0	0	0	0	94.02

Table 5.4. Confusion matrix for 1D CapsNetB on ModelNet-10 (values are in %)

	Predicted										
Actual	Objects	Bathtub	Bed	Chair	Desk	Dresser	Monitor	Night Stand	Sofa	Table	Toilet
	Bathtub	95.24	0	0.12	0	0	0	0	0	0	4.64
	Bed	0	92.45	0	0	0	0	0	5.19	2.36	0
	Chair	0	0	93.77	2.01	0	0	0	1.06	0	3.16
	Desk	0	0	0	87.68	4.6	0	6.52	0	1.2	0
	Dresser	0	0	0	4.16	84.4	0	7.9	0	3.54	0
	Monitor	0	0	0	0	0	93.47	0	0	6.53	0
	Night Stand	0	0	0	5.85	6.99	0	86.11	0	1.05	0
	Sofa	0	0	10.67	0.47	0.53	0	0	88.33	0	0
	Table	0	0	0.65	3.01	1.16	0	1.99	0	93.19	0
	Toilet	2.23	0	1.95	0	0	0	0	0	0.06	95.76

The same class table, in Table 5.4, is confused with chair, desk, dresser and nightstand. In both cases, toilet, which is commode, shows the highest level of accuracy (according to the percentage it acquired). However, many of the commodes are also wrongly classified as bathtubs which can be attributed to the similarities in their shape. Therefore, objects which have quite similar shapes can confuse the neural networks. Dresser had the lowest true positive prediction percentage. The dresser is confused with desk, night stand and table as they all are very similar in shape. Moreover, in the case of 1D CapsNetA, chair is the only object that is confused with the toilet (commode). In the case of 1D CapsNetB, the monitor is the only object that it is confused with the table.

5.5. Conclusion

The work presented in this chapter shows that the difficult task of 3D object recognition with good levels of accuracy, fewer trainable parameters, and using smaller datasets is achievable through the 1D CapsNets architectures which differ only in the Max-Pool layer. These architectures use point cloud dataset to train the parameters. This is achieved through a combination of convolutional layers, a Max-Pool layer, and capsules which help to extract the information directly from the point cloud dataset. Overall, the highest level of 3D object recognition can be performed by the neural networks, 1D CapsNets, which are more efficient, leaner, and less complicated.

The designed 1D CapsNetA and 1D CapsNetB can detect only one object at a time. This helps an IAS to identify objects in any orientation as required. Now that the means of identifying objects in any orientation has been developed, the next chapter concentrates on the development of a more efficient indoor home scene recognition system using CNN. The developed technique in this chapter can then be used in conjunction with the CNN scene recognition architecture to assist IAS systems in identifying indoor areas and locate objects as required.

Chapter – 6

A CNN Combination for Scene Recognition and Conjunction with 1D CapsNet for General Object Detection

6.1. Introduction

In chapter 4, using NoSquashCapsNet, indoor home scene recognition was developed but the accuracy produced, although adequate for many tasks, it is not the best it could be achieved. However, the same is true with other techniques that were compared as there was not much difference in accuracy between NoSquashCapsNet and the other approaches. In this chapter, a CNN system is used to affect an indoor home scene identification (e.g., bedroom, living room, etc.) through multiple object recognition, all in their expected orientations.

In this chapter, section 6.2 discusses the main reasons behind the object detection for scene recognition. Section 6.3 is about the development of pretrained Mask-RCNN and CNN combination for indoor home scene recognition using combinations of objects available in the scenes. It should be noted that the objects are expected to be in their intended orientation as this combination will not deal with varying objects orientations. Section 6.4 discusses the produced outputs by the developed CNN combination. Section 6.5 concludes the chapter.

6.2 Reason behind using object recognition for indoor home scene recognition

Literature shows that, scene and object recognition tasks are considered different tasks, and for this reason, they have always been treated as separate entities (Chapter 3, section 3.3.3, 3.3.4, 3.3.5 and 3.3.6). The works that have implemented scene recognition using object detection have either used techniques like adaptive object detection (Chapter 3, section 3.3.5) [107, 108] or have only partially used neural networks [137, 138]. The partial use of neural networks can be understood form the work presented in [137] in which Fast-RCNN [97] is used to first detect objects in different scenes. Then using the results obtained from Fast-RCNN, two techniques COOR (co-occurring frequency of object-to-object relation) and SOOR (sequential representation of object-to-object relation) are developed to establish object-to-

object relationship. The established object relationships using COOR are then used to train SVM (support vector machine) for scene classification, whereas, in case of SOOR, first the established object relationships are encoded using RNN (Recurrent Neural Network) and then MLP (Multi-Layer Perceptron) is used for scene classification. This whole technique still produced very low accuracy (from 50% to 66.9% on different datasets).

It can be argued that scene recognition is more complex than object recognition. In object recognition, a neural network has to learn only one object at a time, whereas in a scene, a neural network has to learn a complete scene as a whole. In other words, in learning scenes, systems have to learn all the patterns within the image of a scene without special consideration for objects present. Further, in scene recognition tasks, similarities between different scenes make the learning process for a neural network difficult. Naturally, the number and type of patterns present in scenes are more complex than they are for individual objects.

Water Bottle

Living Room Scene

Figure 6.1. An image of a water bottle and a living room

This could be easily understood from the water bottle and the living room scene images shown in Figure 6.1. From Figure 6.1, it is clear that a neural network, when shown the image of the water bottle, has to learn the patterns of only the water bottle, but when the neural network is shown the image of a living room then it has to learn the patterns associated with the concept of a living room (i.e., combined pattern of sofa, armchairs, fireplace, etc). However, it also includes in the learning process the patterns of other things that are not directly relevant, such as the patterns of windows, for example. Clearly such generic pattern combinations can also be present in other home scenes, thus making the task of establishing the distinction between different indoor home scenes very challenging. As a result, while attempting to recognise indoor home scenes, a neural network could confuse one indoor home scene with another if the overall patterns in the scenes are close. However, the situations occurring in a scene do not occur when learning to recognise individual objects. Learning objects to recognize scene could be done using instantaneous segmentation which is discussed

in section 6.3. Nonetheless, indoor home scene evaluation through recognition of associated objects in the scene using only neural networks has remained mostly unexplored. Yet, this appears to be a natural way of achieving high levels of accuracy of scene recognition since the presence of certain objects usually betrays the use of a home room or area. Therefore, in this chapter (section 6.3), a pretrained Mask-RCNN [103] is used for multiple object detection and then a CNN is connected to the output of the Mask-RCNN to predict the scene as per the output of the Mask-RCNN.

6.3. Pretrained Mask-RCNN and CNN combination

In this section, a pre-trained Mask-RCNN is used to perform multiple object detection in 2D. The motivation behind using a pre-trained Mask-RCNN is to utilise the concept of transfer learning [139]. Mask-RCNN has been discussed in Chapter 3, section 3.3.6. The architecture of the Mask-RCNN is shown in Figure 6.2.

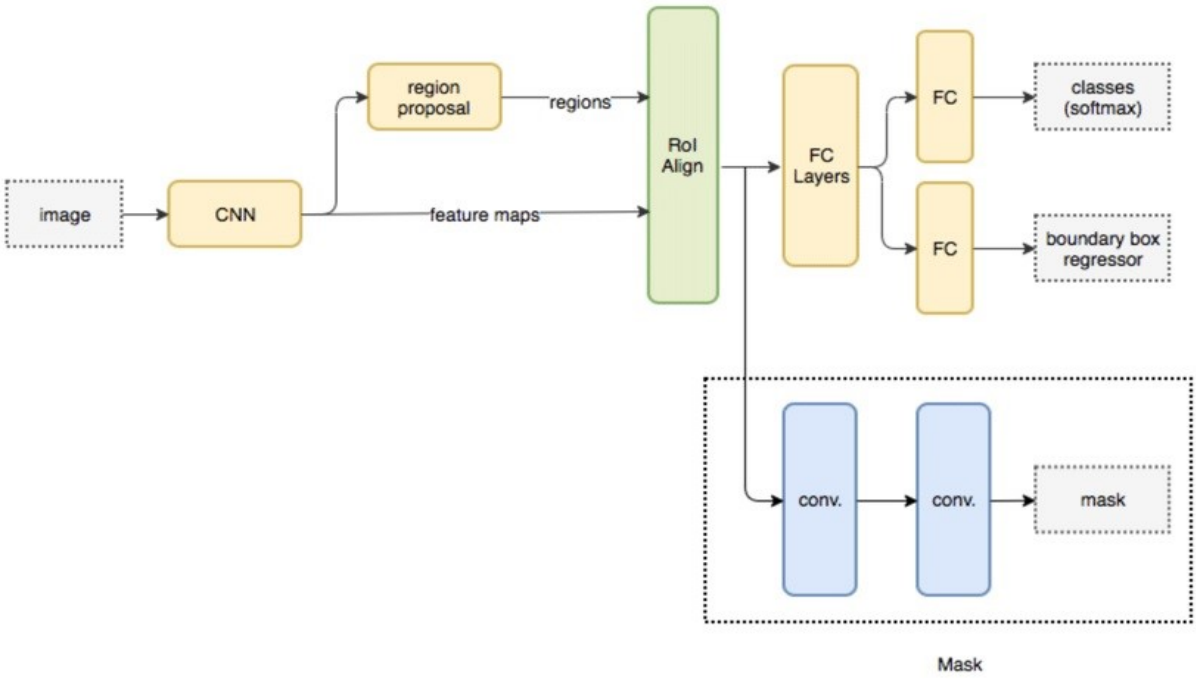


Figure 6.2. Mask-RCNN Architecture [103]

The mask present in the Mask-RCNN enables the CNN to perform instance segmentation which helps to recognise all objects, different or similar, separately. The Instance segmentation performed by the pre-trained Mask-RCNN on the COCO dataset is the main reason behind using Mask-RCNN. This helps in extracting the information of all the objects present in an indoor home scene.

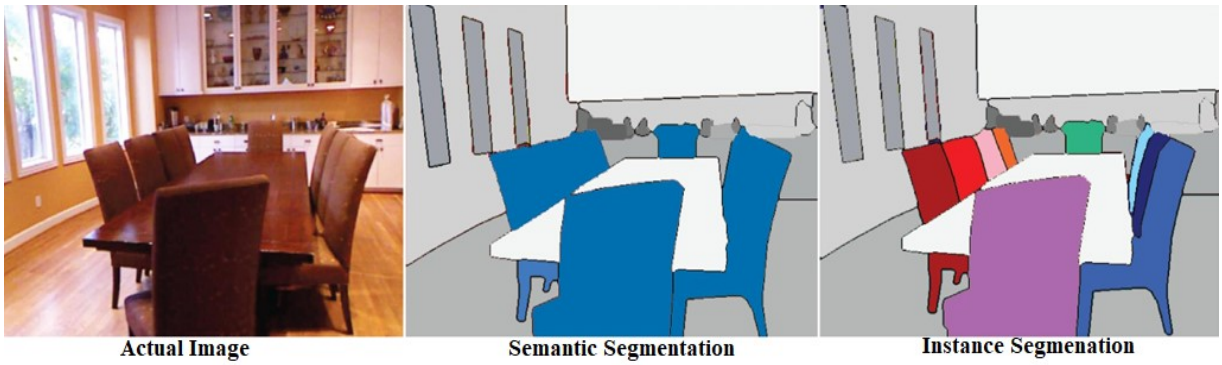


Figure 6.3. A dining room scene showing instance segmentation [162]

The instance segmentation can be better understood from Figure 6.3, in which a dining room scene is shown. In instance segmentation, it can be seen clearly that all objects present in the scene are recognised separately, whereas in semantic segmentation, similar objects are recognised as a single entity. Therefore, for accurate indoor home scene recognition through object detection instance segmentation is important so that all objects present are recognised separately, which, in turn, helps a neural network to properly learn the combination of objects within a scene. Overall, the Mask-RCNN performs an identification of objects in 500 different images of bathrooms, bedrooms, dining rooms, living rooms and kitchens. From Places365 dataset 500 indoor home scenes (100 images for each of the 5 scenes) are extracted.

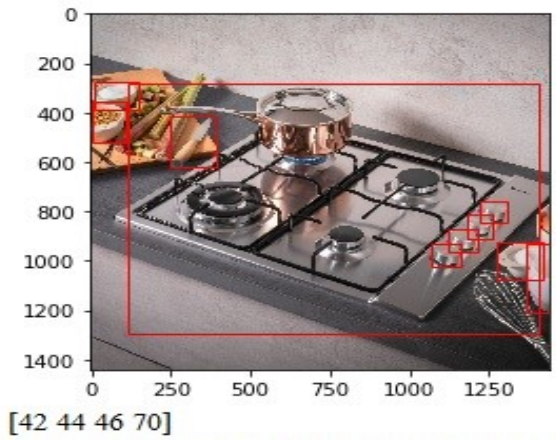


Figure 6.4. Objects in kitchen recognised by Mask-RCNN

Figure 6.4, represents an output produced by Mask-RCNN. In Figure 6.4, the Mask-RCNN has recognised objects associated with a kitchen. The recognised objects are represented in lists as, for example, [42, 44, 46, 70], where 42, 44, 46 and 70 are cup, knife, bowl and cooker respectively. In this way, the pre-trained Mask-RCNN is used to produce 100 combinations of various objects for each indoor home scene category (bathroom, bedroom, kitchen, living room and dining room). The 500 combinations obtained are converted into a dataset. Mask-RCNN produces the output in a sorted form. The object detected by Mask-RCNN are always in

ascending order which can be seen in Figures 6.4, 6.8, 6.9, 6.10, 6.11 and 6.12. Each combination is provided with a label, either bathroom, bedroom, dining room, living room, or kitchen. The CNN is then trained using the obtained labelled data. For training and validation, 80% and 20% of data are used, respectively. The process can be seen in Figure 6.5.

Figure 6.5. Training process for indoor home scene recognition using object detection

The CNN used in this process is one-dimensional because the obtained data is one-dimensional. The architecture of the developed CNN is shown in Figure 6.6. The reason behind not using transfer learning for implementing a CNN is because the approach of using the indoor home dataset obtained from a Mask-RCNN to train a CNN has never been used before. Although there are CNNs pretrained on various objects, there are no CNNs available which have been trained on combinations of objects which is how this dataset has been generated. Therefore, this proves the novelty of the work presented in this chapter as it uses a CNN to evaluate scenes form combinations of objects generated by another CNN (in this case the Mask-RCNN). Hence, this approach establishes a technique to perform scene recognition from object recognition using only neural networks.

Before selecting the shown CNN architecture in Figure 6.6, many different architectures were tried and tested. CNN architectures with 1, 2, 4, 5 and 6 convolutional layers were used. When the convolutional layers used were either 1 or 2, the accuracy produced was around 55%. When the number of convolutional layers was increased to 4, 5 or 6, then the accuracy was also found to be between 50-60%. Pooling layers were also used to improve the performance of the mentioned architectures but the accuracy remained low. Along with the increase and decrease in the number of convolutional layers, fully connected layers with varying numbers of neurons were also tested. However, making variations in fully connected layers also did not help to improve CNN's performance on the required dataset.

The CNN shown in Figure 6.6 was also tried and tested using different numbers of filters and different filter sizes for convolutional layers resulting in CNN producing only 40-50% of accuracy. Therefore, after much experimentation, the CNN architecture shown in Figure 6.6 was chosen as it produced good results as compared to other aforementioned architectures.

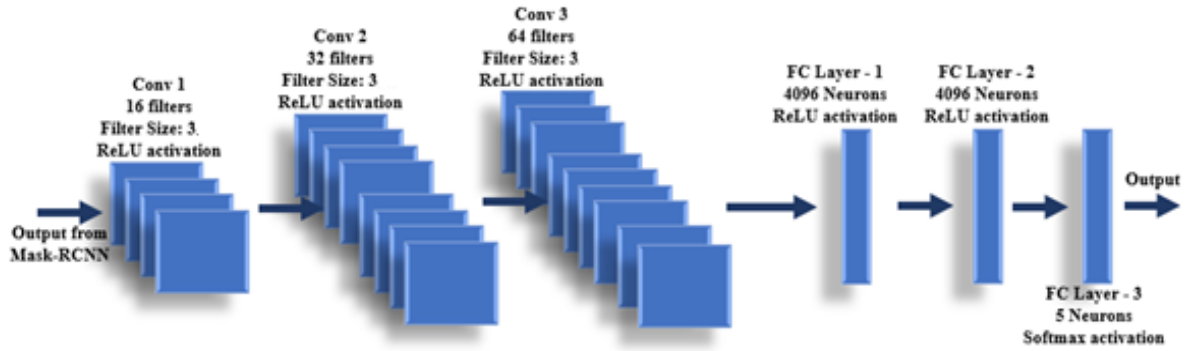


Figure 6.6. CNN developed for Scene Recognition

Table 6.1. CNN architecture specifications

Layers	No. of Filters/Neurons	Filter Size	Activation Function
Conv1	16 filters	3	ReLU
Conv2	32 filters	3	ReLU
Conv3	64 filters	3	ReLU
FC layer-1	4096 neurons	-	ReLU
FC layer-2	4096 neurons	-	ReLU
FC layer-3	n-outputs (5)	-	Softmax

As shown in Table 6.1, the CNN architecture contains three 1D convolutional layers: Conv1, Conv2, and Conv3. All Conv layers have the same filter size of 3. The number of filters for Conv1, Conv2, and Conv3 is 16, 32, and 64, respectively. All Conv layers have ReLU (Rectified Linear Unit) activation function, and no padding has been used in any Conv layer. There stride is 1 for all convolutional layers. After Conv3 layer, there are three FC (Fully Connected) layers. The first and second FC layers have 4096 neurons with ReLU activation function. The third FC layer has neurons equal to the output, which is 5, with a softmax function.

The combination of the pre-trained Mask-RCNN and the trained CNN used for testing is shown in Figure 6.7. From the Places365 dataset [105], 24000 images are used for testing. The images are evenly distributed among the bathroom, bedroom, dining room, living room and kitchen. As already indicated, the object combinations are produced by the Mask-RCNN, and these then become the input for the CNN which, in turn, predicts the indoor home scene.

Figure 6.7. Working of Mask-RCNN and CNN

6.4. Results for Mask-RCNN + CNN combination

After training, the proposed structure produced an accuracy of 97.14%, as indicated in Table 6.2. When compared to other works on scene recognition this is the highest accuracy reported whilst also using more scenes (5 in total) than other methods. The reason behind higher accuracy is, the CNN (which identifies indoor home scenes based on combinations of objects generated by Mask-RCNN) is trained on a statistical dataset that represents different recognised object combinations for different scenes. The task performed by the proposed technique is faster and much simpler than other mentioned techniques. This is because of the very low number of trainable parameters of CNN (10,000 -15,000 approx.), a very small dataset to train, and the use of transfer learning (pre-trained Mask-RCNN). Therefore, learning 500 object combinations (produced by Mask-RCNN) by CNN takes only 30 minutes when trained using 4 Nvidia 1080Ti GPU (Graphic Processor Units). Further, the problem of the presence of similar objects in different indoor home scenes is eliminated because, in this technique, objects present in the scene are used to recognise the scenes, instead of dealing with the scene as a whole (discussed in section 6.1, Figure 6.1).

The accuracies produced by EfficientNet in Afif et al. [115] are 95.6% on the MIT67 dataset [9] (bathroom, bedroom, kitchen, and living room) and 97% on Scene 15 dataset (bedroom, kitchen, and living room). CLM (CodeBookless Model) [112] tested on the same Scene 15 dataset [114] like Afif et al. [115] produced 90% accuracy. However, other than the proposed structure, all the mentioned techniques in this paragraph were trained on mixed indoor scenes (i.e., the dataset included other indoor scenes along with indoor home scenes). Like the EfficientNet and CLM are trained MIT67 dataset [9], which is a mixed scene indoor dataset and then tested only on indoor home scenes of MIT67 dataset.

Table 6.2. Comparison of accuracies of different neural networks used for scene recognition

Neural Network	Accuracy (%)
Proposed Method (Mask-RCNN + CNN) (5 home scenes)	**97.14**
Afif et al. EfficientNet (4 home scenes) [115]	97
Afif et al. EfficientNet (3 home scenes) [115]	95.6
ImageNet-GoogLeNet [105]	96.13
Places365-VGG [105]	92.99
Hybrid1365-VGG [105]	92.15
CLM [112]	90
Places401-Deeper-BN-Inception [106]	86.7
Unified-CNN [104]	51.7

Some other neural networks which produced good accuracies on indoor scenes (this includes indoor home scenes along with other indoor scenes) are ImageNet-GoogLeNet with 96.13% on Event8 dataset, Places365-VGG with 92.99% on SUN attribute dataset [140], Hybrid1365-VGG with 92.15% on Scene 15 dataset, and Places401-Deeper-BN-Inception with 86.7% on MIT67 dataset [9]. Unified-CNN produced an accuracy of 51.7% in scene recognition tasks, whereas it had 52.7% accuracy for object detection. All mentioned neural networks are CNNs trained on very large datasets to acquire the highest possible accuracy. The mentioned techniques in this paragraph, are trained and tested on mixed scene datasets like the Places365 dataset [105] which has millions of scene images of both outdoor and indoor. However, as already mentioned, Places401-Deeper-BN-Inception is tested on MIT67 dataset [9], which is a mixed indoor scene dataset.

Table 6.3. Confusion matrix for indoor home scene recognition by designed CNN

		Predicted				
Actual	**Scenes**	**Bathroom**	**Bedroom**	**Dining Room**	**Kitchen**	**Living Room**
	Bathroom	**98%**	0.5%	0%	0.9%	0.6%
	Bedroom	0%	**98.1%**	0%	1.9%	0%
	Dining Room	0%	0%	**94.25%**	5.75%	0%
	Kitchen	0%	0%	1.8%	**98.2%**	0%
	Living Room	0%	0%	2.85%	0%	**97.15%**

A confusion matrix was obtained to examine the class-wise indoor home scene prediction done by the CNN, as shown in Table 6.3. The green cells of the table are true positive results, i.e., the green cells show the percentage of accurate prediction for each class. The red cells show the false-positive results, i.e., the percentage of wrong predictions. The wrong classification might have happened because of the insufficient presence of objects in respective scenes or they could be because of high number of object similarities between scenes.

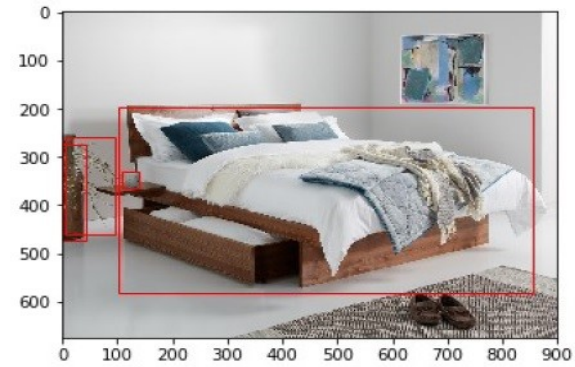
The correctness of the developed method is checked by using random images. In indoor home scenes, similarities and often duplication of objects exist. Hence, the method is tested if it can clearly distinguish between indoor home scenes despite such similarities. Figure 6.8 is the image of a living room. There are red rectangles in the living room image that indicate the

objects detected in the living room scene. The objects detected in Figure 6.8 are represented as [57 58 59 74 76], where 57, 58, 59, 74 and 76 represent a chair, couch, potted plant, book and vase, respectively.


```
[57 58 59 74 76]
[[1.7224460e-08 8.5565279e-12 7.7887211e-04 2.5235431e-07 9.9922085e-01]]
[4]
livingroom
```

Figure 6.8. Scene recognition through object recognition in living room

The numbers below the object combination list in Figure 6.8 represent the probabilities of the scenes in order, [bathroom bedroom dining-room kitchen living-room] which are assigned the numbers [0 1 2 3 4]. In this case, the highest probability is given to the living-room by the CNN, represented as "[4]" - which indicates a correct prediction. In Figure 6.8, there are objects like a flower vase, chair and books in the living room scene. These are the objects which can be found in any other indoor home scene. Still, the CNN responsible for predicting the indoor home scene using the object combination correctly predicts the indoor home scene.




```
[42 59 60 76]
[[1.2231063e-10 1.0000000e+00 2.2445845e-09 1.0146228e-11 6.8124009e-12]]
[1]
bedroom
```

Figure 6.9. Scene recognition through object recognition in bedroom

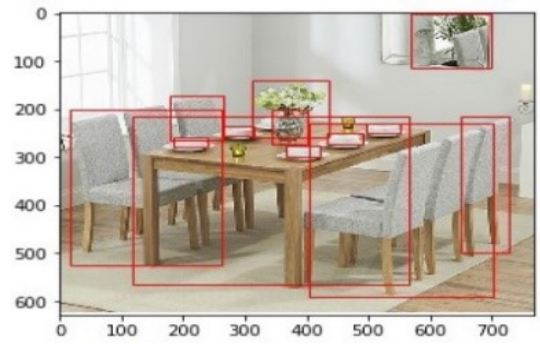
Figure 6.9 is the image of a bedroom. In the bedroom image, there are red rectangles that indicate the objects detected. The objects detected in Figure 6.9 are represented as [42 59 60

76], where 42, 59, 60 and 76 represent cup, potted plant, bed and vase, respectively. Below the object combination numbers in Figure 6.9 represent the probabilities of the scenes. The CNN produces highest probability for bedroom and produces "[1]" as output, which is a bedroom. Therefore, the prediction is correct.

[40 59 62 72 76]
[[9.9999952e-01 2.2787933e-15 1.2968398e-20 4.2241561e-07 2.3824250e-13]]
[0]
bathroom

Figure 6.10 Scene recognition through object recognition in bathroom

Figure 6.10 is the image of a bathroom. In the bathroom image, there are red rectangles that indicate the objects detected. The objects detected in Figure 6.10 are represented as [40 59 62 72 76], where 40, 59, 62, 72 and 76 represent bottle, potted plant, toilet, sink and vase, respectively. Below the object combination numbers in Figure 6.10 represent the probabilities of the scenes. The highest probability is given to the bathroom in this case by CNN, represented as "[0]". The prediction is correct. In Figure 6.10, an object like bottle, can be found in any indoor home scene, is detected in the bathroom scene. This shows that CNN has learnt the object combinations associated with the bathroom correctly.



[46 57 59 61 76]
[[4.4095223e-06 1.7862092e-03 9.6281660e-01 5.1074539e-04 3.4882016e-02]]
[2]
diningroom

Figure 6.11. Scene recognition through object recognition in the dining room

Figure 6.11 represents a dining room. In the image the red rectangles represent the objects detected by the Mask-RCNN. The detected objects are represented as [46 57 59 61 76] where, 46, 57, 59, 61 and 76 represent bowl, chair, potted plant, dining table and vase respectively. On the basis of predicted object combination by the Mask-RCNN, CNN predicts the scene as "[2]" which is a dining room – a correct prediction. The scene is predicted correctly despite the presence of objects like bowl and potted plant, where, a bowl can also be in a kitchen and a potted plant can be present anywhere in the home. This shows that the CNN has been able to learn the exact combination of objects that constitute a dining room.

Figure 6.12. Scene recognition through object recognition in kitchen

Figure 6.12 is the image of a kitchen. In the kitchen image, there are red rectangles that indicate the objects detected. The objects detected in Figure 6.12 are represented as [1 40 64 69 70 73 75], where 1, 40, 64, 69, 70, 73 and 75 represent background objects, bottle, laptop, microwave, oven, refrigerator and clock, respectively. Below the object combination numbers in Figure 6.12 represent the probabilities of the scenes. The highest probability is given to the dining room in this case by CNN, which is represented as "[3]". The prediction is correct. The thing to be noted in Figure 6.12 is that objects like laptop and wall clock are detected. A laptop is an unusual object to be found in the kitchen. Moreover, a wall clock is an object that can be found in any indoor home scene. Still, CNN, through the obtained object combination, detects the scene properly. This shows that CNN has learnt the exact object combination associated with the kitchen. The results discussed show that a robust system for indoor home scene recognition has been developed.

6.5. Conclusion

This chapter presented indoor home scene recognition technique through object recognition using only neural networks. The developed CNN combination technique produced the best accuracy and helped in solving the problem of high similarity between different indoor home scenes. This will enable an IAS to perform an indoor home scene and object recognition tasks smoothly. The obtained high degree of accuracy illustrates the effectiveness of the proposed scheme.

Chapter – 7

Conclusion and Future Works

7.1 Conclusion

This thesis addresses issues associated with indoor home scene and object recognition related to IAS (Intelligent Assistive Systems) which are designed to assist elderly and/or infirm people by way of accomplishing tasks associated with indoor activities and/or locating and transporting objects. New and novel approaches for indoor home scene recognition and object recognition using different deep learning techniques have been developed and their performance has been evaluated and compared to other existing methods. The thesis concentrates on different CapsNets (Capsule Neural Networks) architectures which have been developed and used to perform efficient scene and object recognition separately. These different CapsNets architectures are the NoSquashCapsNet used for indoor home scene recognition, and the 1D CapsNets used for 3D indoor object recognition. Further improvement in indoor home scene recognition was achieved by combining a purposely designed CNN architecture with a Mask-RCNN existing design to produce very high accuracy in indoor scene recognition. Finally, the 1D CapsNet and the combined Mask-RCNN + CNN structures are merged so that an IAS can use them together to accurately identify the indoor area surrounding it and any objects which are required and which may be in any orientation.

Chapter 2 of this thesis describes, based on the literature review, why there is a need for assistive systems. Further, it goes on to outline different existing indoor systems designed to assist elderly and infirm people. These indoor systems involve mobile systems available (section 2.3.1), different fall detection systems (section 2.3.2) and different systems available for navigation in indoor home environments (section 2.3.3). The review of different indoor systems helped to understand the need for developing efficient scene and object recognition techniques for assistive systems. The review also helped to understand why deep learning techniques will be the most appropriate for developing efficient techniques for IAS.

An extensive literature review on different techniques available for scene and object detection is provided in Chapter 3. Furthermore, chapter 3 also explains, in some detail, the basics of CNN and CapsNet. Most of the techniques discussed are based on deep learning techniques implemented for scene and object detection. The review of different techniques

helps to understand that, in the case of indoor home scene recognition, there is not a wide availability of methods which are trained and tested only on indoor home scenes. The idea of CapsNet was explored in an attempt to perform both scene and object recognition using just one system. CapsNet is ideal for 3D object recognition because of its ability to learn the orientation of objects at no extra computational cost. Another research area which was important to explore was the use of only deep learning techniques to perform scene recognition using object detection. This topic not only had remained largely unexplored, but the techniques used for scene recognition using object recognition had proved not very efficient and produced low accuracy. The review helped to understand that there is lack of efficient learning techniques. The available deep learning approaches are not capable of producing higher accuracy when trained on smaller datasets and also do not have lower number of trainable parameters even though they are trained on smaller datasets.

In Chapter 4, an indoor home scene recognition is developed using CapsNet. The traditional CapsNet is implemented for indoor home scene recognition in section 4.2.5. The performance of the CapsNet is compared with Mask-RCNN, Fast RCNN and Faster RCNN. The CapsNet was able to produce an accuracy which is comparable to the aforementioned methods but somewhat below the accuracies of Fast-RCNN and Faster-RCNN (section 4.2.7, Table 4.2). Nonetheless when the training dataset was reduced to 5000 images, the CapsNet accuracy was not affected. By contrast, the accuracy of all the other neural networks dropped substantially. The ability of the CapsNet to retain the accuracy despite the substantial reduction in the amount of data within the dataset showed that CapsNets can produce better accuracy on smaller datasets (section 4.2.7, Table 4.3). However, CapsNet had too many trainable parameters, over 160 million. The higher number of trainable parameters of CapsNet possibly aid to reach a saturation level but no conclusive evidence is available for the same. Therefore, this led to the development of NoSquashCapsNet (section 4.3.2, Figure 4.7). In the NoSquashCapsNet architecture, capsule without squash function is used and pooling layers are introduced in CapsNet architecture. Squash function restricts the direction of vectors from being changed. Removing squash helps in removing that restriction. Introduction of Max-Pool layers in the architecture helps in further mitigating the problem of high numbers of trainable parameters. NoSquashCapsNet produced equivalent accuracy to that of traditional CapsNet. However, NoSquashCapsNet proves to be more efficient than traditional CapsNet because it has total number of trainable parameters of 30 million+ (section 4.3.3, Table 4.9). The confusion matrices (Table 4.4, 4.5, 4.6 and 4.10) also helped to understand that capturing a scene image

as a whole (learning the patterns of the whole scene) confuses the neural networks, resulting in lower accuracy. Overall, then, in scene recognition (e.g., a room) orientation is not a requirement as the main objects which dictate the type of scene are always as expected (tables, chairs, etc,). As a result, in this case, the orientation advantage of CapsNet-based architecture is not so important and it was deduced that this architecture did not offer much improvement compared to other methods, apart from not requiring large datasets for a comparable performance. An improved method of obtaining accurate scene recognition is presented in Chapter 6 where indoor home scene recognition is performed using object recognition.

The knowledge acquired on CapsNets in Chapter 4 helped to further develop CapsNets for 3D object recognition in Chapter 5. There are two CapsNets developed in Chapter 5, 1D CapsNetA and 1D CapsNetB (section 5.3, Figure 5.4). The reason behind developing 1D CapsNets is to utilise the point cloud dataset so that the total number of parameters remains low by avoiding the conversion of the point cloud dataset to 3D voxel grids (section 5.2). Further, the development of 1D CapsNet enables to have better accuracy on smaller dataset. The reasons behind producing better accuracy on smaller dataset are, (a) the information supplied is in 3D and, (b) the CapsNet's ability to capture objects' orientation. The idea of using point cloud dataset is derived from the PointNet (Section 5.2). PointNet became the basis of further development of better architectures. However, PointNet has a complicated architecture. Therefore, this led to the development of PointCapsNet architecture (Figure 5.3). When compared to PointNet, PointCapsNet produced better accuracy on ModelNet-10 but a drop in the case of ModelNet-40 indicating that this was a better architecture for 3D object recognition. However, the problems present in PointNet still existed in PointCapsNet like learning an objects' orientation using T-Nets and presence of combination of different neural networks which did not help with either training or further performance improvement (Section 5.2). Therefore, this led to the development of the 1D CapsNet architectures.

The difference between the two 1D CapsNet architectures is only the position of the Max-Pool layer. On both datasets, 1D CapsNetB produced the better accuracy (91.04% on ModelNet-40 and 92.03% on ModelNet-10) compared to the 1D CapsNetA (Table 5.2). 1D CapsNet has one of the best performances but it has less total number of trainable parameters as compared to other methods (Table 5.2). Both 1D CapsNets have substantially reduced trainable parameters because: (a). Capsules are used in 1D, (b). the data in 1D array format and (c). because of the introduction of Max-Pool layer. Hence, it can be concluded that efficient architectures for object recognition have been developed with better accuracy despite having

less trainable parameters and being trained on a smaller dataset. The best part about the developed architectures is that they are capable of recognising the objects presented to them in any orientation because of the properties of the capsule (section 4.2.3, 4.2.4 and 5.3) and the use of 3D datasets for training.

It can be concluded from Chapter 4 and 5 that CapsNets are very efficient in learning the orientations but are less efficient in learning scene arrangements. Moreover, from the newly designed CapsNet architectures (NoSquashCapsNet, 1D CapsNetA and 1D CapsNetB) it is clear that CapsNets performed well when pooling layers were integrated into them. Further, it is also shown that CapsNets architectures with lower trainable parameters can achieve better accuracy.

Chapter 6 of this thesis illustrates scene recognition through multiple object detection and how to combine the scene recognition with object recognition irrespective of object orientation. The pre-trained Mask-RCNN is chosen to perform the object recognition because it can perform instance segmentation and it was readily available through transfer learning (section 6.3). Object recognition was performed using pre-trained Mask-RCNN on 500 different indoor home scenes. This helped to generate a unique dataset which represented different objects recognized in a particular scene (section 6.3, Figure 6.4). A CNN was then developed which can be trained on the generated dataset to perform the indoor home scene recognition (section 6.3, Figure 6.6). After training the CNN, its input is connected with the output of the pre-trained Mask-RCNN (section 6.4, Figure 6.5). The developed combination was tested on 24000 different indoor home scenes and produced the highest accuracy compared to other existing techniques (section 6.4, Table 6.2). Therefore, it can be concluded that a complete system has been developed which solves the problem of accurate and efficient indoor home scene and object recognition in IAS.

The results obtained for indoor home scene and object recognition show that, in this thesis, efficient techniques with better accuracies have been developed which can use smaller datasets for training and which have less total number of trainable parameters and, therefore, they require less training time and effort. A unique development has also been produced which uses only neural networks to perform indoor home scene recognition through object recognition. This development helps to substantially reduce problems in indoor home scene recognition such as requirements of large datasets, slow training process, use of same scene and object images from different angles, etc. These are the advantages which can help IAS to perform the tasks much faster and with substantially less effort. Furthermore, the developed techniques now

enable an IAS to move within an indoor home environment more efficiently, through the use of the Mask-RCNN+CNN method. At the same time an IAS can also recognise any object present in any orientation using 1D CapsNet. However, further work could be carried out to improve the developed techniques presented in this thesis. This is discussed in the next section.

7.2. Future Works

There are many aspects associated with the work presented in the thesis that could be explored further,

- In Chapter 4, it is seen that the accuracy of CapsNets did not drop when the dataset was decreased but the accuracy also did not improve when the data in the dataset was increased which potentially points to a saturation level of CapsNets. This needs further investigation to reach a conclusive reason for why this happens in CapsNets. If the reason behind accuracy saturation becomes known, it may help to further improve the performance of CapsNets. The same applies to the 1D CapsNets presented in Chapter 5.

- In Chapter 5, developed 1D CapsNets can recognise only one object at a time. Therefore, as it stands, the present development is incapable of performing multiple object detection like the Mask-RCNN shown in Chapter 6. Should CapsNet become capable of performing instance segmentation and semantic segmentation. CapsNet can be developed in way such that it is able to show the objects recognised using bounding boxes. This can help to replace the Mask-RCNN in our architecture.

- In Chapter 6, combination of CNNs to perform indoor home scene recognition using object recognition. Instead of having a combination of neural networks to perform the required task, single neural network must be developed that can itself perform indoor home scene recognition by recognising the objects present in the scene. This could replace the developed Mask-RCNN+CNN combination. However, this will require development of a completely new neural network that can perform two tasks which presently does not exist.

- Datasets are the most important single part in deep learning for creating an efficient system. Therefore, the above-mentioned future works can be implemented using the 3D indoor environment dataset developed by Matterport and recently made available for

academic use. This would help to replace the dataset used in this thesis with a much better dataset which could help in producing improved results.

- The conjunction of Mask-RCNN+CNN (Chapter 6) and 1D CapsNet (Chapter 5) can be implemented so that the indoor navigation of an IAS can be improved further. This can be implemented in real time (for example, in an existing mobile robot with computer vision capabilities). (Appendix D)

Appendix

Appendix A – Python Code for NoSquashCapsNet (Referred to Chapter 4)

A. I. NoSquashCapsNet

```
"""
Created on Fri May 10 11:59:09 2019

@author: gpu-server

Github: `https://github.com/XifengGuo/CapsNet-Keras` (reference)
"""
import numpy as np
from keras import layers, models, optimizers
from keras import backend as K
from keras.utils import to_categorical
import matplotlib.pyplot as plt
from utils import combine_images
from PIL import Image
from capsulelayersVer3 import CapsuleLayer, PrimaryCap, Length, Mask,
PrimaryCapWithoutReshape1
from capsule_layers import ConvCapsuleLayer
from keras.utils import multi_gpu_model
#from keras.layers.advanced_activations import LeakyReLU
#from keras.layers.advanced_activations import PReLU
from keras.layers import Dropout
from keras.layers.merge import concatenate
from keras.layers.normalization import BatchNormalization

K.set_image_data_format('channels_last')

def CapsNet(input_shape, n_class, routings):

    x = layers.Input(shape=input_shape)


    # Layer 1: Just a conventional Conv2D layer
    conv1_1 = layers.Conv2D(filters=64, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv1_1')(x)
    conv1_2 = layers.Conv2D(filters=64, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv1_2')(x)
    conv1_3 = layers.Conv2D(filters=64, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv1_3')(x)
    concat1 = concatenate([conv1_1, conv1_2, conv1_3], axis=3)
    pool1 = layers.MaxPooling2D(pool_size=(2, 2), strides=2,
padding='same', data_format=None)(concat1)

    conv2_1 = layers.Conv2D(filters=128, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv2_1')(pool1)
    conv2_2 = layers.Conv2D(filters=128, kernel_size=3, strides=1,
```

```
padding='same', activation= 'relu', name='conv2_2')(pool1)
    conv2_3 = layers.Conv2D(filters=128, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv2_3')(pool1)
    concat2 = concatenate([conv2_1, conv2_2, conv2_3], axis=3)
    pool2 = layers.MaxPooling2D(pool_size=(2, 2), strides=2,
padding='same', data_format=None)(concat2)

    conv3_1 = layers.Conv2D(filters=256, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv3_1')(pool2)
    conv3_2 = layers.Conv2D(filters=256, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv3_2')(pool2)
    conv3_3 = layers.Conv2D(filters=256, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv3_3')(pool2)

    concat3 = concatenate([conv3_1, conv3_2, conv3_3], axis=3)

    pool3 = layers.MaxPooling2D(pool_size=(2, 2), strides=2,
padding='same', data_format=None)(concat3)

    conv4_1 = layers.Conv2D(filters=512, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv4_1')(pool3)
    conv4_2 = layers.Conv2D(filters=512, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv4_2')(pool3)
    conv4_3 = layers.Conv2D(filters=512, kernel_size=3, strides=1,
padding='same', activation= 'relu', name='conv4_3')(pool3)

    concat4 = concatenate([conv4_1, conv4_2, conv4_3], axis=3)

    primarycaps = PrimaryCapWithoutReshape1(concat4, dim_capsule=32,
n_channels=16, kernel_size=3, strides=2, padding='valid')


    # Layer 3: Capsule layer. Routing algorithm works here.
    digitcaps = CapsuleLayer(num_capsule=n_class, dim_capsule=16,
routings=routings,
                             name='digitcaps')(primarycaps)


    # Layer 4: This is an auxiliary layer to replace each capsule with its
length. Just to match the true label's shape.
    # If using tensorflow, this will not be necessary. :)
    out_caps = Length(name='capsnet')(digitcaps)

    # Decoder network.
    y = layers.Input(shape=(n_class,))
    masked_by_y = Mask()([digitcaps, y])  # The true label is used to mask
the output of capsule layer. For training
    masked = Mask()(digitcaps)  # Mask using the capsule with maximal
length. For prediction


    # Shared Decoder model in training and prediction
    decoder = models.Sequential(name='decoder')
    decoder.add(layers.Dense(4096, activation='relu',
input_dim=16*n_class))
#     decoder.add(BatchNormalization())
#     decoder.add(Dropout(0.5))
    decoder.add(layers.Dense(4096, activation='relu'))
#     decoder.add(BatchNormalization())
#     decoder.add(Dropout(0.5))
    decoder.add(layers.Dense(5, activation='relu'))
```

```
    decoder.add(layers.Dense(np.prod(input_shape), activation='sigmoid'))
    decoder.add(layers.Reshape(target_shape=input_shape, name='out_recon'))



    # Models for training and evaluation (prediction)
    print(x.shape)
    print(y.shape)
    print(out_caps.shape)
    print(decoder(masked_by_y).shape)
    train_model = models.Model([x, y], [out_caps, decoder(masked_by_y)])
    eval_model = models.Model(x, [out_caps, decoder(masked)])

    # manipulate model
    noise = layers.Input(shape=(n_class, 16))
    noised_digitcaps = layers.Add()([digitcaps, noise])
    masked_noised_y = Mask()([noised_digitcaps, y])
    manipulate_model = models.Model([x, y, noise],
decoder(masked_noised_y))
    return train_model, eval_model, manipulate_model




def margin_loss(y_true, y_pred):
    """
    Margin loss for Eq.(4). When y_true[i, :] contains not just one `1`,
this loss should work too. Not test it.
    :param y_true: [None, n_classes]
    :param y_pred: [None, num_capsule]
    :return: a scalar loss value.
    """
    L = y_true * K.square(K.maximum(0., 0.9 - y_pred)) + \
        0.5 * (1 - y_true) * K.square(K.maximum(0., y_pred - 0.1))

    return K.mean(K.sum(L, 1))


def train(model, data, args):
    """
    Training a CapsuleNet
    :param model: the CapsuleNet model
    :param data: a tuple containing training and testing data, like
`((x_train, y_train), (x_test, y_test))`
    :param args: arguments
    :return: The trained model
    """
    # unpacking the data
    (x_train, y_train), (x_test, y_test) = data
#     (x_train, y_train) = data

    # callbacks
    log = callbacks.CSVLogger(args.save_dir + '/log.csv')
    tb = callbacks.TensorBoard(log_dir=args.save_dir + '/tensorboard-logs',
                               batch_size=args.batch_size,
histogram_freq=int(args.debug))
    checkpoint = callbacks.ModelCheckpoint(args.save_dir + '/weights-
{epoch:02d}.h5', monitor='val_capsnet_acc',
                                           save_best_only=True,
save_weights_only=True, verbose=1)
    lr_decay = callbacks.LearningRateScheduler(schedule=lambda epoch:
```

```
args.lr * (args.lr_decay ** epoch))

    # compile the model
    model.compile(optimizer=optimizers.Adam(lr=args.lr),
                  loss=[margin_loss, 'mse'],
#                  loss='categorical_crossentropy',
                  loss_weights=[1., args.lam_recon],
                  metrics={'capsnet': 'accuracy'})


#    # Training without data augmentation:
    model.fit([x_train, y_train], [y_train, x_train],
batch_size=args.batch_size, epochs=args.epochs,
              validation_data=[[x_test, y_test], [y_test, x_test]],
callbacks=[log, tb, checkpoint, lr_decay])

    # Begin: Training with data augmentation -----------------------------
-------------------------------------#
#    def train_generator(x, y, batch_size, shift_fraction=0.):
#        train_datagen =
ImageDataGenerator(width_shift_range=shift_fraction,
#
height_shift_range=shift_fraction)  # shift up to 2 pixel for MNIST
#        generator = train_datagen.flow(x, y, batch_size=batch_size)
#        while 1:
#            x_batch, y_batch = generator.next()
#            yield ([x_batch, y_batch], [y_batch, x_batch])
#
#    # Training with data augmentation. If shift_fraction=0., also no
augmentation.
#    model.fit_generator(generator=train_generator(x_train, y_train,
args.batch_size, args.shift_fraction),
#                        steps_per_epoch=int(y_train.shape[0] /
args.batch_size),
#                        epochs=args.epochs,
#                        validation_data=[[x_test, y_test], [y_test,
x_test]],
#                        callbacks=[log, tb, checkpoint, lr_decay])
    # End: Training with data augmentation -----------------------------
-------------------------------------#

    model.save_weights(args.save_dir + '/trained_model.h5')
    print('Trained model saved to \'%s/trained_model.h5\'' % args.save_dir)

    from utils import plot_log
    plot_log(args.save_dir + '/log.csv', show=True)

    return model


def test(model, data, args):
    x_test, y_test = data
    y_pred, x_recon = model.predict(x_test, batch_size=100)
    print('-'*30 + 'Begin: test' + '-'*30)
    print('Test acc:', np.sum(np.argmax(y_pred, 1) == np.argmax(y_test,
1))/y_test.shape[0])

    img = combine_images(np.concatenate([x_test[:50],x_recon[:50]]))
    image = img * 255
    Image.fromarray(image.astype(np.uint8)).save(args.save_dir +
"/real_and_recon.png")
```

```
    print()
    print('Reconstructed images are saved to %s/real_and_recon.png' %
args.save_dir)
    print('-' * 30 + 'End: test' + '-' * 30)
    plt.imshow(plt.imread(args.save_dir + "/real_and_recon.png"))
    plt.show()


def manipulate_latent(model, data, args):
    print('-'*30 + 'Begin: manipulate' + '-'*30)
    x_test, y_test = data
    index = np.argmax(y_test, 1) == args.digit
    number = np.random.randint(low=0, high=sum(index) - 1)
    x, y = x_test[index][number], y_test[index][number]
    x, y = np.expand_dims(x, 0), np.expand_dims(y, 0)
    noise = np.zeros([1, 10, 16])
    x_recons = []
    for dim in range(16):
        for r in [-0.25, -0.2, -0.15, -0.1, -0.05, 0, 0.05, 0.1, 0.15, 0.2,
0.25]:
            tmp = np.copy(noise)
            tmp[:,:,dim] = r
            x_recon = model.predict([x, y, tmp])
            x_recons.append(x_recon)

    x_recons = np.concatenate(x_recons)

    img = combine_images(x_recons, height=16)
    image = img*255
    Image.fromarray(image.astype(np.uint8)).save(args.save_dir +
'/manipulate-%d.png' % args.digit)
    print('manipulated result saved to %s/manipulate-%d.png' %
(args.save_dir, args.digit))
    print('-' * 30 + 'End: manipulate' + '-' * 30)


#def load_mnist():
#     # the data, shuffled and split between train and test sets
#     from keras.datasets import mnist
#     (x_train, y_train), (x_test, y_test) = mnist.load_data()
#
#     x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.
#     x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.
#     y_train = to_categorical(y_train.astype('float32'))
#     y_test = to_categorical(y_test.astype('float32'))
#     return (x_train, y_train), (x_test, y_test)
#
def load_images():

    images=np.load('npz_data.npy')

    labels=np.load('npz_label_data.npy')

    num_example=images.shape[0]
    ratio=0.9
    s=np.int(num_example*ratio)

    x_train = images[:s]
    y_train = labels[:s]
    x_test = images[s:]
    y_test = labels[s:]
```

```
    x_train = x_train.reshape(-1, 128, 128, 3).astype('float32') / 255.
    x_test = x_test.reshape(-1, 128, 128, 3).astype('float32') / 255.
    y_train = y_train.reshape(-1, 5).astype('float32')
    y_test = y_test.reshape(-1, 5).astype('float32')

    #label=labels.reshape([-1, 10]).astype('float32')

    return (x_train, y_train), (x_test, y_test)

if __name__ == "__main__":
    import os
    import argparse
    from keras.preprocessing.image import ImageDataGenerator
    from keras import callbacks

    # setting the hyper parameters
    parser = argparse.ArgumentParser(description="Capsule Network on
MNIST.")
    parser.add_argument('--epochs', default=100, type=int)
    parser.add_argument('--batch_size', default=100, type=int)
    parser.add_argument('--lr', default=0.0001, type=float,
                        help="Initial learning rate")
    parser.add_argument('--lr_decay', default=0.9, type=float,
                        help="The value multiplied by lr at each epoch. Set
a larger value for larger epochs")
    parser.add_argument('--lam_recon', default=0.392, type=float,
                        help="The coefficient for the loss of decoder")
    parser.add_argument('-r', '--routings', default=3, type=int,
                        help="Number of iterations used in routing
algorithm. should > 0")
    parser.add_argument('--shift_fraction', default=0.1, type=float,
                        help="Fraction of pixels to shift at most in each
direction.")
    parser.add_argument('--debug', action='store_true',
                        help="Save weights by TensorBoard")
    parser.add_argument('--save_dir', default='./result23')
    parser.add_argument('-t', '--testing', action='store_true',
                        help="Test the trained model on testing dataset")
    parser.add_argument('--digit', default=2, type=int,
                        help="Digit to manipulate")
    parser.add_argument('-w', '--weights', default=None,
                        help="The path of the saved weights. Should be
specified when testing")
    parser.add_argument('--gpus', default=4, type=int)
    args = parser.parse_args()
    print(args)

    if not os.path.exists(args.save_dir):
        os.makedirs(args.save_dir)

    # load data
#    (x_train, y_train), (x_test, y_test) = load_images()
    (x_train, y_train), (x_test, y_test) = load_images()

    print(x_train.shape)
    print(y_train.shape)
    # define model
    model, eval_model, manipulate_model =
CapsNet(input_shape=x_train.shape[1:],
```

```
n_class=len(np.unique(np.argmax(y_train, 1))),
                                                routings=args.routings)
    model.summary()

    # train or test
#    if args.weights is not None:  # init the model weights with provided
one
#        model.load_weights(args.weights)
#    if not args.testing:
#        train(model=model, data=((x_train, y_train), (x_test, y_test)),
args=args)
#    else:  # as long as weights are given, will run testing
#        if args.weights is None:
#            print('No weights are provided. Will test using random
initialized weights.')
#        manipulate_latent(manipulate_model, (x_test, y_test), args)
#        test(model=eval_model, data=(x_test, y_test), args=args)

# train or test
    if args.weights is not None:  # init the model weights with provided
one
        model.load_weights(args.weights)
    if not args.testing:
        # define muti-gpu model
        multi_model = multi_gpu_model(model, gpus=args.gpus)
        train(model=multi_model, data=((x_train, y_train), (x_test,
y_test)), args=args)
        model.save_weights(args.save_dir + '/trained_model.h5')
        print('Trained model saved to \'%s/trained_model.h5\'' %
args.save_dir)
        test(model=eval_model, data=(x_test, y_test), args=args)
    else:  # as long as weights are given, will run testing
        if args.weights is None:
            print('No weights are provided. Will test using random
initialized weights.')
        manipulate_latent(manipulate_model, (x_test, y_test), args)
        test(model=eval_model, data=(x_test, y_test), args=args)
```

A. II. Capsule without squash function

```
"""
Created on Mon Mar 11 12:38:35 2019

@author: kkb17226
"""

"""
Github: `https://github.com/XifengGuo/CapsNet-Keras` (reference)
"""

import keras.backend as K
import tensorflow as tf
from keras import initializers, layers
class Length(layers.Layer):
    """
    Compute the length of vectors. This is used to compute a Tensor that
```

```
has the same shape with y_true in margin_loss.
    Using this layer as model's output can directly predict labels by using
`y_pred = np.argmax(model.predict(x), 1)`
    inputs: shape=[None, num_vectors, dim_vector]
    output: shape=[None, num_vectors]
    """
    def call(self, inputs, **kwargs):
        return K.sqrt(K.sum(K.square(inputs), -1) + K.epsilon())

    def compute_output_shape(self, input_shape):
        return input_shape[:-1]

    def get_config(self):
        config = super(Length, self).get_config()
        return config

class Mask(layers.Layer):
        def call(self, inputs, **kwargs):
        if type(inputs) is list:  # true label is provided with shape =
[None, n_classes], i.e. one-hot code.
            assert len(inputs) == 2
            inputs, mask = inputs
        else:  # if no true label, mask by the max length of capsules.
Mainly used for prediction
            # compute lengths of capsules
            x = K.sqrt(K.sum(K.square(inputs), -1))
            # generate the mask which is a one-hot code.
            # mask.shape=[None, n_classes]=[None, num_capsule]
            mask = K.one_hot(indices=K.argmax(x, 1),
num_classes=x.get_shape().as_list()[1])

        # inputs.shape=[None, num_capsule, dim_capsule]
        # mask.shape=[None, num_capsule]
        # masked.shape=[None, num_capsule * dim_capsule]
        masked = K.batch_flatten(inputs * K.expand_dims(mask, -1))
        return masked

    def compute_output_shape(self, input_shape):
        if type(input_shape[0]) is tuple:  # true label provided
            return tuple([None, input_shape[0][1] * input_shape[0][2]])
        else:  # no true label provided
            return tuple([None, input_shape[1] * input_shape[2]])

    def get_config(self):
        config = super(Mask, self).get_config()
        return config


class CapsuleLayer(layers.Layer):
    """
    The capsule layer. It is similar to Dense layer. Dense layer has
`in_num` inputs, each is a scalar, the output of the
    neuron from the former layer, and it has `out_num` output neurons.
CapsuleLayer just expand the output of the neuron
    from scalar to vector. So its input shape = [None, input_num_capsule,
input_dim_capsule] and output shape = \
    [None, num_capsule, dim_capsule]. For Dense Layer, input_dim_capsule =
dim_capsule = 1.

    :param num_capsule: number of capsules in this layer
    :param dim_capsule: dimension of the output vectors of the capsules in
```

```
this layer
    :param routings: number of iterations for the routing algorithm
    """
    def __init__(self, num_capsule, dim_capsule, routings=3,
                 kernel_initializer='glorot_uniform',
                 **kwargs):
        super(CapsuleLayer, self).__init__(**kwargs)
        self.num_capsule = num_capsule
        self.dim_capsule = dim_capsule
        self.routings = routings
        self.kernel_initializer = initializers.get(kernel_initializer)

    def build(self, input_shape):
        assert len(input_shape) >= 3, "The input Tensor should have
shape=[None, input_num_capsule, input_dim_capsule]"
        self.input_num_capsule = input_shape[1]
        self.input_dim_capsule = input_shape[2]

        # Transform matrix
        self.W = self.add_weight(shape=[self.num_capsule,
self.input_num_capsule,
                                        self.dim_capsule,
self.input_dim_capsule],
                                 initializer=self.kernel_initializer,
                                 name='W')

        self.built = True

    def call(self, inputs, training=None):
        # inputs.shape=[None, input_num_capsule, input_dim_capsule]
        # inputs_expand.shape=[None, 1, input_num_capsule,
input_dim_capsule]
        inputs_expand = K.expand_dims(inputs, 1)

        # Replicate num_capsule dimension to prepare being multiplied by W
        # inputs_tiled.shape=[None, num_capsule, input_num_capsule,
input_dim_capsule]
        inputs_tiled = K.tile(inputs_expand, [1, self.num_capsule, 1, 1])

        # Compute `inputs * W` by scanning inputs_tiled on dimension 0.
        # x.shape=[num_capsule, input_num_capsule, input_dim_capsule]
        # W.shape=[num_capsule, input_num_capsule, dim_capsule,
input_dim_capsule]
        # Regard the first two dimensions as `batch` dimension,
        # then matmul: [input_dim_capsule] x [dim_capsule,
input_dim_capsule]^T -> [dim_capsule].
        # inputs_hat.shape = [None, num_capsule, input_num_capsule,
dim_capsule]
        inputs_hat = K.map_fn(lambda x: K.batch_dot(x, self.W, [2, 3]),
elems=inputs_tiled)

        # Begin: Routing algorithm ---------------------------------------
------------------------------#
        # The prior for coupling coefficient, initialized as zeros.
        # b.shape = [None, self.num_capsule, self.input_num_capsule].
        b = tf.zeros(shape=[K.shape(inputs_hat)[0], self.num_capsule,
self.input_num_capsule])

        assert self.routings > 0, 'The routings should be > 0.'
        for i in range(self.routings):
            # c.shape=[batch_size, num_capsule, input_num_capsule]
```

```
            c = tf.nn.softmax(b, dim=1)

            # c.shape =  [batch_size, num_capsule, input_num_capsule]
            # inputs_hat.shape=[None, num_capsule, input_num_capsule,
dim_capsule]
            # The first two dimensions as `batch` dimension,
            # then matmal: [input_num_capsule] x [input_num_capsule,
dim_capsule] -> [dim_capsule].
            # outputs.shape=[None, num_capsule, dim_capsule]
            outputs = squash(K.batch_dot(c, inputs_hat, [2, 2]))  # [None,
10, 16]

            if i < self.routings - 1:
                # outputs.shape =  [None, num_capsule, dim_capsule]
                # inputs_hat.shape=[None, num_capsule, input_num_capsule,
dim_capsule]
                # The first two dimensions as `batch` dimension,
                # then matmal: [dim_capsule] x [input_num_capsule,
dim_capsule]^T -> [input_num_capsule].
                # b.shape=[batch_size, num_capsule, input_num_capsule]
                b += K.batch_dot(outputs, inputs_hat, [2, 3])
        # End: Routing algorithm -----------------------------------------
----------------------------#

        return outputs

    def compute_output_shape(self, input_shape):
        return tuple([None, self.num_capsule, self.dim_capsule])

    def get_config(self):
        config = {
            'num_capsule': self.num_capsule,
            'dim_capsule': self.dim_capsule,
            'routings': self.routings
        }
        base_config = super(CapsuleLayer, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))


def PrimaryCapWithoutReshape1(inputs, dim_capsule, n_channels, kernel_size,
strides, padding):
    """
    Apply Conv2D `n_channels` times and concatenate all capsules
    :param inputs: 4D tensor, shape=[None, width, height, channels]
    :param dim_capsule: the dim of the output vector of capsule
    :param n_channels: the number of types of capsules
    :return: output tensor, shape=[None, num_capsule, dim_capsule]
    """
    output1 = layers.Conv2D(filters=dim_capsule*n_channels,
kernel_size=kernel_size, strides=strides, padding=padding,
                            name='primarycapwithoutreshape1_conv2d')(inputs)
        return output1
```

Appendix B – Python Code for 1D CapsNet, 1D Capsule and PointCapsNet (Referred to Chapter 5)

B. I. 1D CapsNet

```
"""
Created on Mon Aug 26 16:36:18 2019

@author: gpu-server

https://github.com/charlesq34/pointnet (reference)
"""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May 10 11:59:09 2019

@author: gpu-server
"""


import numpy as np
from keras import layers, models, optimizers
from keras import backend as K
from keras.utils import to_categorical
import matplotlib.pyplot as plt
from keras.utils import multi_gpu_model
from keras.layers import Dropout
from keras.layers.merge import concatenate
from keras.layers.normalization import BatchNormalization
import os
import tensorflow as tf
from keras import layers, models
from keras import optimizers
from keras.layers import Input
from keras.models import Model
from keras.layers import Dense, Flatten, Reshape, Dropout
from keras.layers import Convolution1D, MaxPooling1D, BatchNormalization
from keras.layers import Lambda
from keras.utils import np_utils
import h5py
from capsulelayer1D import CapsuleLayer, PrimaryCapWithoutReshape1, Length,
Mask

def mat_mul(A, B):
    return tf.matmul(A, B)


def load_h5(h5_filename):
    f = h5py.File(h5_filename)
```

```
    data = f['data'][:]
    label = f['label'][:]
    return (data, label)


def rotate_point_cloud(batch_data):
    """ Randomly rotate the point clouds to augument the dataset
        rotation is per shape based along up direction
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, rotated batch of point clouds
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, 0, sinval],
                                    [0, 1, 0],
                                    [-sinval, 0, cosval]])
        shape_pc = batch_data[k, ...]
        rotated_data[k, ...] = np.dot(shape_pc.reshape((-1, 3)),
rotation_matrix)
    return rotated_data


def jitter_point_cloud(batch_data, sigma=0.01, clip=0.05):
    """ Randomly jitter points. jittering is per point.
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, jittered batch of point clouds
    """
    B, N, C = batch_data.shape
    assert(clip > 0)
    jittered_data = np.clip(sigma * np.random.randn(B, N, C), -1 * clip,
clip)
    jittered_data += batch_data
    return jittered_data

num_points = 1024

# number of categories
k = 10

# define optimizer
adam = optimizers.Adam(lr=0.001, decay=0.7)

input_points = Input(shape=(num_points, 3))

    # Layer 1: Just a conventional Conv2D layer
conv1_1 = layers.Conv1D(64, 1, activation='relu', input_shape=(num_points,
3), name='conv1_1')(input_points)
conv1_2 = layers.Conv1D(64, 1, activation= 'relu', input_shape=(num_points,
3), name='conv1_2')(input_points)
#conv1_3 = layers.Conv1D(64, 1, activation= 'relu',
input_shape=(num_points, 3), name='conv1_3')(input_points)
concat1 = concatenate([conv1_1, conv1_2], axis=1)
#pool1 = layers.MaxPooling1D(pool_size=2048)(concat1)
```

```
conv2_1 = layers.Conv1D(128, 1, activation= 'relu',
name='conv2_1')(concat1)
conv2_2 = layers.Conv1D(128, 1, activation= 'relu',
name='conv2_2')(concat1)
#conv2_3 = layers.Conv1D(128, 1, activation= 'relu',
name='conv2_3')(concat1)
concat2 = concatenate([conv2_1, conv2_2], axis=1)
#pool2 = layers.MaxPooling1D(pool_size=2048)(concat2)

pool3 = layers.MaxPooling1D(pool_size=2048)(concat2)

conv3_1 = layers.Conv1D(256, 1, activation= 'relu', name='conv3_1')(pool3)
conv3_2 = layers.Conv1D(256, 1, activation= 'relu', name='conv3_2')(pool3)
#conv3_3 = layers.Conv1D(256, 1, activation= 'relu', name='conv3_3')(pool2)

concat3 = concatenate([conv3_1, conv3_2], axis=1)

#pool3 = layers.MaxPooling1D(pool_size=2048)(concat3)

conv4_1 = layers.Conv1D(512, 1, activation= 'relu',
name='conv4_1')(concat3)
conv4_2 = layers.Conv1D(512, 1, activation= 'relu',
name='conv4_2')(concat3)
#conv4_3 = layers.Conv1D(512, 1, activation= 'relu',
name='conv4_3')(concat3)

concat4 = concatenate([conv4_1, conv4_2], axis=1)
#pool3 = layers.MaxPooling1D(pool_size=2048)(concat4)

primarycaps = PrimaryCapWithoutReshape1(concat4, dim_capsule=32,
n_channels=16, kernel_size=1)

digitcaps = CapsuleLayer(num_capsule=k, dim_capsule=16, routings=3,
                         name='digitcaps')(primarycaps)

out_caps = Length(name='capsnet')(digitcaps)

 # Decoder network.
y = layers.Input(shape=(k,))
masked_by_y = Mask()([digitcaps, y])  # The true label is used to mask the
output of capsule layer. For training
masked = Mask()(digitcaps)  # Mask using the capsule with maximal length.
For prediction

# Shared Decoder model in training and prediction
decoder = models.Sequential(name='decoder')
decoder.add(layers.Dense(4096, activation='relu', input_dim=16*k))
#     decoder.add(BatchNormalization())
#     decoder.add(Dropout(0.5))
decoder.add(layers.Dense(4096, activation='relu'))
#     decoder.add(BatchNormalization())
#     decoder.add(Dropout(0.5))
decoder.add(layers.Dense(5, activation='relu'))
decoder.add(layers.Dense(k, activation='softmax'))


# print the model summary

model = Model(inputs=input_points, outputs = out_caps)
print(model.summary())
```

```
try:
    model = multi_gpu_model(model)
except:
    pass

# load train points and labels
path = os.path.dirname(os.path.realpath('./cap1D/'))
train_path = os.path.join(path, "PrepData")
filenames = [d for d in os.listdir(train_path)]
print(train_path)
print(filenames)
train_points = None
train_labels = None
for d in filenames:
    cur_points, cur_labels = load_h5(os.path.join(train_path, d))
    cur_points = cur_points.reshape(1, -1, 3)
    cur_labels = cur_labels.reshape(1, -1)
    if train_labels is None or train_points is None:
        train_labels = cur_labels
        train_points = cur_points
    else:
        train_labels = np.hstack((train_labels, cur_labels))
        train_points = np.hstack((train_points, cur_points))
train_points_r = train_points.reshape(-1, num_points, 3)
train_labels_r = train_labels.reshape(-1, 1)

# load test points and labels
test_path = os.path.join(path, "PrepData_test")
filenames = [d for d in os.listdir(test_path)]
print(test_path)
print(filenames)
test_points = None
test_labels = None
for d in filenames:
    cur_points, cur_labels = load_h5(os.path.join(test_path, d))
    cur_points = cur_points.reshape(1, -1, 3)
    cur_labels = cur_labels.reshape(1, -1)
    if test_labels is None or test_points is None:
        test_labels = cur_labels
        test_points = cur_points
    else:
        test_labels = np.hstack((test_labels, cur_labels))
        test_points = np.hstack((test_points, cur_points))
test_points_r = test_points.reshape(-1, num_points, 3)
test_labels_r = test_labels.reshape(-1, 1)


# label to categorical
Y_train = np_utils.to_categorical(train_labels_r, k)
Y_test = np_utils.to_categorical(test_labels_r, k)

# compile classification model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit model on training data
for i in range(1,12):
    #model.fit(train_points_r, Y_train, batch_size=32, epochs=1,
shuffle=True, verbose=1)
    # rotate and jitter the points
```

```
    train_points_rotate = rotate_point_cloud(train_points_r)
    train_points_jitter = jitter_point_cloud(train_points_rotate)
    model.fit(train_points_jitter, Y_train, batch_size=32, epochs=1,
shuffle=True, verbose=1)
    s = "Current epoch is:" + str(i)
    print(s)
    if i % 5 == 0:
        score = model.evaluate(test_points_r, Y_test, verbose=1)
        print('Test loss: ', score[0])
        print('Test accuracy: ', score[1])

# score the model
score = model.evaluate(test_points_r, Y_test, verbose=1)
print('Test loss: ', score[0])
print('Test accuracy: ', score[1])
```

B. II. 1D Capsule

```
"""
Created on Mon Aug 26 16:44:03 2019

@author: gpu-server
"""

"""
Created on Mon Mar 11 12:38:35 2019

@author: gvb17226
"""

"""
`https://github.com/XifengGuo/CapsNet-Keras`(reference)
"""

import keras.backend as K
import tensorflow as tf
from keras import initializers, layers


class Length(layers.Layer):
    """
    Compute the length of vectors. This is used to compute a Tensor that
has the same shape with y_true in margin_loss.
    Using this layer as model's output can directly predict labels by using
`y_pred = np.argmax(model.predict(x), 1)`
    inputs: shape=[None, num_vectors, dim_vector]
    output: shape=[None, num_vectors]
    """
    def call(self, inputs, **kwargs):
        return K.sqrt(K.sum(K.square(inputs), -1) + K.epsilon())

    def compute_output_shape(self, input_shape):
        return input_shape[:-1]

    def get_config(self):
        config = super(Length, self).get_config()
```

```
        return config


class Mask(layers.Layer):
    """
    Mask a Tensor with shape=[None, num_capsule, dim_vector] either by the
capsule with max length or by an additional
    input mask. Except the max-length capsule (or specified capsule), all
vectors are masked to zeros. Then flatten the
    masked Tensor.
    For example:
        ```
        x = keras.layers.Input(shape=[8, 3, 2]) # batch_size=8, each
sample contains 3 capsules with dim_vector=2
        y = keras.layers.Input(shape=[8, 3]) # True labels. 8 samples, 3
classes, one-hot coding.
        out = Mask()(x) # out.shape=[8, 6]
        # or
        out2 = Mask()([x, y]) # out2.shape=[8,6]. Masked with true labels
y. Of course y can also be manipulated.
        ```
    """
    def call(self, inputs, **kwargs):
        if type(inputs) is list:  # true label is provided with shape =
[None, n_classes], i.e. one-hot code.
            assert len(inputs) == 2
            inputs, mask = inputs
        else:  # if no true label, mask by the max length of capsules.
Mainly used for prediction
            # compute lengths of capsules
            x = K.sqrt(K.sum(K.square(inputs), -1))
            # generate the mask which is a one-hot code.
            # mask.shape=[None, n_classes]=[None, num_capsule]
            mask = K.one_hot(indices=K.argmax(x, 1),
num_classes=x.get_shape().as_list()[1])

        # inputs.shape=[None, num_capsule, dim_capsule]
        # mask.shape=[None, num_capsule]
        # masked.shape=[None, num_capsule * dim_capsule]
        masked = K.batch_flatten(inputs * K.expand_dims(mask, -1))
        return masked

    def compute_output_shape(self, input_shape):
        if type(input_shape[0]) is tuple:  # true label provided
            return tuple([None, input_shape[0][1] * input_shape[0][2]])
        else:  # no true label provided
            return tuple([None, input_shape[1] * input_shape[2]])

    def get_config(self):
        config = super(Mask, self).get_config()
        return config


def squash(vectors, axis=-1):
    """
    The non-linear activation used in Capsule. It drives the length of a
large vector to near 1 and small vector to 0
    :param vectors: some vectors to be squashed, N-dim tensor
    :param axis: the axis to squash
    :return: a Tensor with same shape as input vectors
    """
```

```
    s_squared_norm = K.sum(K.square(vectors), axis, keepdims=True)
    scale = s_squared_norm / (1 + s_squared_norm) / K.sqrt(s_squared_norm +
K.epsilon())
    return scale * vectors


class CapsuleLayer(layers.Layer):
    """
    The capsule layer. It is similar to Dense layer. Dense layer has
`in_num` inputs, each is a scalar, the output of the
    neuron from the former layer, and it has `out_num` output neurons.
CapsuleLayer just expand the output of the neuron
    from scalar to vector. So its input shape = [None, input_num_capsule,
input_dim_capsule] and output shape = \
    [None, num_capsule, dim_capsule]. For Dense Layer, input_dim_capsule =
dim_capsule = 1.

    :param num_capsule: number of capsules in this layer
    :param dim_capsule: dimension of the output vectors of the capsules in
this layer
    :param routings: number of iterations for the routing algorithm
    """
    def __init__(self, num_capsule, dim_capsule, routings=3,
                 kernel_initializer='glorot_uniform',
                 **kwargs):
        super(CapsuleLayer, self).__init__(**kwargs)
        self.num_capsule = num_capsule
        self.dim_capsule = dim_capsule
        self.routings = routings
        self.kernel_initializer = initializers.get(kernel_initializer)

    def build(self, input_shape):
        assert len(input_shape) >= 3, "The input Tensor should have
shape=[None, input_num_capsule, input_dim_capsule]"
        self.input_num_capsule = input_shape[1]
        self.input_dim_capsule = input_shape[2]

        # Transform matrix
        self.W = self.add_weight(shape=[self.num_capsule,
self.input_num_capsule,
                                        self.dim_capsule,
self.input_dim_capsule],
                                 initializer=self.kernel_initializer,
                                 name='W')

        self.built = True

    def call(self, inputs, training=None):
        # inputs.shape=[None, input_num_capsule, input_dim_capsule]
        # inputs_expand.shape=[None, 1, input_num_capsule,
input_dim_capsule]
        inputs_expand = K.expand_dims(inputs, 1)

        # Replicate num_capsule dimension to prepare being multiplied by W
        # inputs_tiled.shape=[None, num_capsule, input_num_capsule,
input_dim_capsule]
        inputs_tiled = K.tile(inputs_expand, [1, self.num_capsule, 1, 1])

        # Compute `inputs * W` by scanning inputs_tiled on dimension 0.
        # x.shape=[num_capsule, input_num_capsule, input_dim_capsule]
        # W.shape=[num_capsule, input_num_capsule, dim_capsule,
```

```
input_dim_capsule]
        # Regard the first two dimensions as `batch` dimension,
        # then matmul: [input_dim_capsule] x [dim_capsule,
input_dim_capsule]^T -> [dim_capsule].
        # inputs_hat.shape = [None, num_capsule, input_num_capsule,
dim_capsule]
        inputs_hat = K.map_fn(lambda x: K.batch_dot(x, self.W, [2, 3]),
elems=inputs_tiled)

        # Begin: Routing algorithm ---------------------------------------
----------------------------#
        # The prior for coupling coefficient, initialized as zeros.
        # b.shape = [None, self.num_capsule, self.input_num_capsule].
        b = tf.zeros(shape=[K.shape(inputs_hat)[0], self.num_capsule,
self.input_num_capsule])

        assert self.routings > 0, 'The routings should be > 0.'
        for i in range(self.routings):
            # c.shape=[batch_size, num_capsule, input_num_capsule]
            c = tf.nn.softmax(b, dim=1)

            # c.shape =  [batch_size, num_capsule, input_num_capsule]
            # inputs_hat.shape=[None, num_capsule, input_num_capsule,
dim_capsule]
            # The first two dimensions as `batch` dimension,
            # then matmal: [input_num_capsule] x [input_num_capsule,
dim_capsule] -> [dim_capsule].
            # outputs.shape=[None, num_capsule, dim_capsule]
            outputs = squash(K.batch_dot(c, inputs_hat, [2, 2]))  # [None,
10, 16]

            if i < self.routings - 1:
                # outputs.shape =  [None, num_capsule, dim_capsule]
                # inputs_hat.shape=[None, num_capsule, input_num_capsule,
dim_capsule]
                # The first two dimensions as `batch` dimension,
                # then matmal: [dim_capsule] x [input_num_capsule,
dim_capsule]^T -> [input_num_capsule].
                # b.shape=[batch_size, num_capsule, input_num_capsule]
                b += K.batch_dot(outputs, inputs_hat, [2, 3])
        # End: Routing algorithm -----------------------------------------
----------------------------#

        return outputs

    def compute_output_shape(self, input_shape):
        return tuple([None, self.num_capsule, self.dim_capsule])

    def get_config(self):
        config = {
            'num_capsule': self.num_capsule,
            'dim_capsule': self.dim_capsule,
            'routings': self.routings
        }
        base_config = super(CapsuleLayer, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))


def PrimaryCapWithoutReshape1(inputs, dim_capsule, n_channels,
kernel_size):
    """
```

```
    Apply Conv2D `n_channels` times and concatenate all capsules
    :param inputs: 4D tensor, shape=[None, width, height, channels]
    :param dim_capsule: the dim of the output vector of capsule
    :param n_channels: the number of types of capsules
    :return: output tensor, shape=[None, num_capsule, dim_capsule]
    """
    output1 = layers.Conv1D(filters=dim_capsule*n_channels,
kernel_size=kernel_size,
                            name='primarycapwithoutreshape1_conv2d')(inputs)
#    output2 = layers.MaxPooling1D(pool_size=2048)(output1)
#    outputs = layers.Reshape(target_shape=[-1, dim_capsule],
name='primarycap_reshape')(output1)
    return layers.Lambda(squash,
name='primarycapwithoutreshape1_squash')(output1)
#    return output1
```

B. III. PointCapsNet

```
"""
Created on Thu Dec 28 15:39:40 2017
Pointnet+Caps
@author: gvb17226
https://github.com/charlesq34/pointnet (reference)
"""

import numpy as np
import os
import tensorflow as tf
from keras import layers, models
from keras import optimizers
from keras.layers import Input
from keras.models import Model
from keras.layers import Dense, Flatten, Reshape, Dropout
from keras.layers import Convolution1D, MaxPooling1D, BatchNormalization
from keras.layers import Lambda
from keras.utils import np_utils
import h5py
from capsulelayer1D import CapsuleLayer, PrimaryCapWithoutReshape1, Length,
Mask


def mat_mul(A, B):
    return tf.matmul(A, B)


def load_h5(h5_filename):
    f = h5py.File(h5_filename)
    data = f['data'][:]
    label = f['label'][:]
    return (data, label)


def rotate_point_cloud(batch_data):
    """ Randomly rotate the point clouds to augument the dataset
        rotation is per shape based along up direction
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, rotated batch of point clouds
```

```
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, 0, sinval],
                                    [0, 1, 0],
                                    [-sinval, 0, cosval]])
        shape_pc = batch_data[k, ...]
        rotated_data[k, ...] = np.dot(shape_pc.reshape((-1, 3)),
rotation_matrix)
    return rotated_data


def jitter_point_cloud(batch_data, sigma=0.01, clip=0.05):
    """ Randomly jitter points. jittering is per point.
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, jittered batch of point clouds
    """
    B, N, C = batch_data.shape
    assert(clip > 0)
    jittered_data = np.clip(sigma * np.random.randn(B, N, C), -1 * clip,
clip)
    jittered_data += batch_data
    return jittered_data


# number of points in each sample
num_points = 2048

# number of categories
k = 10

# define optimizer
adam = optimizers.Adam(lr=0.001, decay=0.7)

# ------------------------------------ Pointnet Architecture
# input_Transformation_net
input_points = Input(shape=(num_points, 3))
x = Convolution1D(64, 1, activation='relu',
                  input_shape=(num_points, 3))(input_points)
x = BatchNormalization()(x)
x = Convolution1D(128, 1, activation='relu')(x)
x = BatchNormalization()(x)
x = Convolution1D(1024, 1, activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling1D(pool_size=num_points)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dense(9, weights=[np.zeros([256, 9]), np.array([1, 0, 0, 0, 1, 0, 0, 0,
1]).astype(np.float32)])(x)
input_T = Reshape((3, 3))(x)

# forward net
g = Lambda(mat_mul, arguments={'B': input_T})(input_points)
g = Convolution1D(64, 1, input_shape=(num_points, 3), activation='relu')(g)
```

```
g = BatchNormalization()(g)
g = Convolution1D(64, 1, input_shape=(num_points, 3), activation='relu')(g)
g = BatchNormalization()(g)

# feature transform net
f = Convolution1D(64, 1, activation='relu')(g)
f = BatchNormalization()(f)
f = Convolution1D(128, 1, activation='relu')(f)
f = BatchNormalization()(f)
f = Convolution1D(1024, 1, activation='relu')(f)
f = BatchNormalization()(f)
f = MaxPooling1D(pool_size=num_points)(f)
f = Dense(512, activation='relu')(f)
f = BatchNormalization()(f)
f = Dense(256, activation='relu')(f)
f = BatchNormalization()(f)
f = Dense(64 * 64, weights=[np.zeros([256, 64 * 64]),
np.eye(64).flatten().astype(np.float32)])(f)
feature_T = Reshape((64, 64))(f)

# forward net
g = Lambda(mat_mul, arguments={'B': feature_T})(g)
g = Convolution1D(64, 1, activation='relu')(g)
g = BatchNormalization()(g)
g = Convolution1D(128, 1, activation='relu')(g)
g = BatchNormalization()(g)
g = Convolution1D(1024, 1, activation='relu')(g)
g = BatchNormalization()(g)

# global_feature
global_feature = MaxPooling1D(pool_size=num_points)(g)

# capsule integration
cap = PrimaryCapWithoutReshape1(global_feature, dim_capsule=32,
n_channels=16, kernel_size=1)
digitcaps = CapsuleLayer(num_capsule=k, dim_capsule=16, routings=3,
                         name='digitcaps')(cap)
out_caps = Length(name='capsnet')(digitcaps)
y = layers.Input(shape=(k,))
masked_by_y = Mask()([digitcaps, y])
masked = Mask()(digitcaps)

# Decoder
decoder = models.Sequential(name='decoder')
decoder.add(layers.Dense(4096, activation='relu', input_dim=16*k))
#     decoder.add(BatchNormalization())
#     decoder.add(Dropout(0.5))
decoder.add(layers.Dense(4096, activation='relu'))
#     decoder.add(BatchNormalization())
#     decoder.add(Dropout(0.5))
decoder.add(layers.Dense(5, activation='relu'))
decoder.add(layers.Dense(k, activation='softmax'))

# ----------------------------------------------------end of pointnet

# print the model summary
model = Model(inputs=input_points, outputs=out_caps)
print(model.summary())

# load train points and labels
path = os.path.dirname(os.path.realpath('./cap1D/'))
```

```
train_path = os.path.join(path, "PrepData")
filenames = [d for d in os.listdir(train_path)]
print(train_path)
print(filenames)
train_points = None
train_labels = None
for d in filenames:
    cur_points, cur_labels = load_h5(os.path.join(train_path, d))
    cur_points = cur_points.reshape(1, -1, 3)
    cur_labels = cur_labels.reshape(1, -1)
    if train_labels is None or train_points is None:
        train_labels = cur_labels
        train_points = cur_points
    else:
        train_labels = np.hstack((train_labels, cur_labels))
        train_points = np.hstack((train_points, cur_points))
train_points_r = train_points.reshape(-1, num_points, 3)
train_labels_r = train_labels.reshape(-1, 1)

# load test points and labels
test_path = os.path.join(path, "PrepData_test")
filenames = [d for d in os.listdir(test_path)]
print(test_path)
print(filenames)
test_points = None
test_labels = None
for d in filenames:
    cur_points, cur_labels = load_h5(os.path.join(test_path, d))
    cur_points = cur_points.reshape(1, -1, 3)
    cur_labels = cur_labels.reshape(1, -1)
    if test_labels is None or test_points is None:
        test_labels = cur_labels
        test_points = cur_points
    else:
        test_labels = np.hstack((test_labels, cur_labels))
        test_points = np.hstack((test_points, cur_points))
test_points_r = test_points.reshape(-1, num_points, 3)
test_labels_r = test_labels.reshape(-1, 1)


# label to categorical
Y_train = np_utils.to_categorical(train_labels_r, k)
Y_test = np_utils.to_categorical(test_labels_r, k)

# compile classification model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit model on training data
for i in range(1,100):
    #model.fit(train_points_r, Y_train, batch_size=32, epochs=1,
shuffle=True, verbose=1)
    # rotate and jitter the points
    train_points_rotate = rotate_point_cloud(train_points_r)
    train_points_jitter = jitter_point_cloud(train_points_rotate)
    model.fit(train_points_jitter, Y_train, batch_size=32, epochs=1,
shuffle=True, verbose=1)
    s = "Current epoch is:" + str(i)
    print(s)
    if i % 5 == 0:
```

```
        score = model.evaluate(test_points_r, Y_test, verbose=1)
        print('Test loss: ', score[0])
        print('Test accuracy: ', score[1])

# score the model
score = model.evaluate(test_points_r, Y_test, verbose=1)
print('Test loss: ', score[0])
print('Test accuracy: ', score[1])
```

Appendix C – Python Code for Mask-RCNN+CNN combination (Referred to Chapter 6)

C. I. Mask-RCNN+CNN combination for indoor home scene recognition using object detection

```
"""
Created on Tue Dec 1 14:59:45 2020

@author: gpu-server

https://github.com/matterport/Mask_RCNN (reference)
"""
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from matplotlib import pyplot
from matplotlib.patches import Rectangle
import pandas as pd
# draw an image with detected objects
def draw_image_with_boxes(filename, boxes_list):
     # load the image
     data = pyplot.imread(filename)
     # plot the image
     pyplot.imshow(data)
     # get the context for drawing boxes
     ax = pyplot.gca()
     # plot each box
     for box in boxes_list:
          # get coordinates
          y1, x1, y2, x2 = box
          # calculate width and height of the box
          width, height = x2 - x1, y2 - y1
          # create the shape
          rect = Rectangle((x1, y1), width, height, fill=False,
color='red')
          # draw the box
          ax.add_patch(rect)
     # show the plot
     pyplot.show()
# define the test configuration
class TestConfig(Config):
     NAME = "test"
     GPU_COUNT = 1
     IMAGES_PER_GPU = 1
     NUM_CLASSES = 1 + 80
# define the model
rcnn = MaskRCNN(mode='inference', model_dir='./', config=TestConfig())
```

```
# load coco model weights
rcnn.load_weights('mask_rcnn_coco.h5', by_name=True)
# load photograph
img = load_img('elephant.jpg')
img = img_to_array(img)
# make prediction
results = rcnn.detect([img], verbose=0)
# visualize the results
draw_image_with_boxes('elephant.jpg', results[0]['rois'])

from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from mrcnn.visualize import display_instances
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
import numpy as np
# define 81 classes that the coco model knowns about
class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
               'bus', 'train', 'truck', 'boat', 'traffic light',
               'fire hydrant', 'stop sign', 'parking meter', 'bench',
'bird',
               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
               'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag',
'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
               'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog',
'pizza',
               'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
               'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote',
               'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
               'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',
               'teddy bear', 'hair drier', 'toothbrush']

# define the test configuration
class TestConfig(Config):
     NAME = "test"
     GPU_COUNT = 1
     IMAGES_PER_GPU = 1
     NUM_CLASSES = 1 + 80

# define the model (pre-trained Mask-RCNN, Transfer Learning)
rcnn = MaskRCNN(mode='inference', model_dir='./', config=TestConfig())
# load coco model weights
rcnn.load_weights('mask_rcnn_coco.h5', by_name=True)


#predict object from 500 rooms and create csv file of object list
table1 = np.zeros((499,81), dtype = int)
for i in range (499):
    i = i+1
    img = load_img('/home/gpu-server/Mask_RCNN/room_data100/room' + str(i)
+ '.jpg')
    path = '/home/gpu-server/Mask_RCNN/room_data100/room' + str(i) + '.jpg'
    img = img_to_array(img)
    results = rcnn.detect([img], verbose=0)
    r = results[0]
    #display_instances(img, r['rois'], r['masks'], r['class_ids'],
class_names, r['scores'])
```

```
    obj = np.unique(r['class_ids'])
    n = len(obj)
    for j in range (n):
        x = obj[j]
        if x > 0:
            table1[i-1,x] = 1
#    for j in range (n):
#    arr = np.reshape(obj, (1,n))
#        table1[i-1,j] = arr[0,j-1]

pd.DataFrame(table1).to_csv("onehot_object500_2.csv")

#Oblect detect for test data
table2 = np.zeros((24,81), dtype = int)
for i in range (24):
    i = i+1
    img = load_img('/home/gpu-server/Mask_RCNN/home_data/room' + str(i) +
'.jpg')
    path = '/home/gpu-server/Mask_RCNN/home_data/room' + str(i) + '.jpg'
    img = img_to_array(img)
    results = rcnn.detect([img], verbose=0)
    r = results[0]
    #display_instances(img, r['rois'], r['masks'], r['class_ids'],
class_names, r['scores'])
    obj = np.unique(r['class_ids'])
    n = len(obj)
    for j in range (n):
        x = obj[j]
        if x > 0:
            table2[i-1,x] = 1
#    arr = np.reshape(obj, (1,n))
#    for j in range (n):
#        table2[i-1,j] = arr[0,j-1]

pd.DataFrame(table2).to_csv("onehot_object.csv")

# cnn model for classify scence from object list
import numpy as np
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
import pandas as pd

trainpath = "/home/gpu-server/Mask_RCNN/dat_w_matlab/onehot_object500.csv"
train_label = "/home/gpu-server/Mask_RCNN/dat_w_matlab/room_label_500.csv"

testpath = "/home/gpu-
server/Mask_RCNN/dat_w_matlab/testing_onehot_object.csv"
test_label = "/home/gpu-server/Mask_RCNN/dat_w_matlab/room.csv"

# load the dataset, returns train and test X and y elements
def load_dataset(trainpath, train_label, testpath, test_label):
```

```
    # load all train
    trainX = pd.read_csv(trainpath)
    trainy = pd.read_csv(train_label)
    print(trainX.shape, trainy.shape)
    # load all test
    testX = pd.read_csv(testpath)
    testy = pd.read_csv(test_label)
    print(testX.shape, testy.shape)
    # zero-offset class values
#   trainy = trainy - 1
#   testy = testy - 1
#   # one hot encode y
#   trainy = to_categorical(trainy)
#   testy = to_categorical(testy)
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return trainX, trainy, testX, testy

#fit and evaluate a model (CNN)
def evaluate_model(trainX, trainy, testX, testy):
    verbose, epochs, batch_size = 1, 100, 10
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    model = Sequential()
    model.add(Conv1D(filters=16, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
    model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
#     model.add(Dropout(0.5))
#     model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save_weights('onehot_room_500.h5')

    # evaluate model
    model.load_weights('onehot_room_500.h5')
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=1)
#     model.summary()
    return accuracy

# summarize scores
def summarize_results(scores):
   print(scores)
   m, s = mean(scores), std(scores)
   print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an experiment
def run_experiment(repeats=1):
    trainX, trainy, testX, testy = load_dataset(trainpath, train_label,
testpath, test_label)
    trainX = np.expand_dims(trainX, axis=2)
    testX = np.expand_dims(testX, axis=2)
   # repeat experiment
    scores = list()
    for r in range(repeats):
```

```
        score = evaluate_model(trainX, trainy, trainX, trainy)
        score = score * 100.0
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # summarize results
    summarize_results(scores)
# run the experiment
run_experiment()
# test with random images
rcnn.load_weights('mask_rcnn_coco.h5', by_name=True)
# load photograph
#img = load_img('kitchen.jpeg')
#img = img_to_array(img)
## make prediction
#results = rcnn.detect([img], verbose=0)
## visualize the results
#draw_image_with_boxes('kitchen.jpeg', results[0]['rois'])
#
#test_obj = np.unique(r['class_ids'])
#print(test_obj)
def prediction_model(test_obj):
    batch_size = 1
    n_timesteps, n_features = test_obj.shape[1], test_obj.shape[2]
    model = Sequential()
    model.add(Conv1D(filters=16, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
    model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.load_weights('onehot_room_500.h5')
    y = model.predict(test_obj, batch_size=batch_size,)
    return y

def run_test(repeats=1):
    table3 = np.zeros((1,81), dtype = int)
    trainX, trainy, testX, testy = load_dataset(trainpath, train_label,
testpath, test_label)
    img = load_img('test9.jpg')
    img = img_to_array(img)
    results = rcnn.detect([img], verbose=0)
    draw_image_with_boxes('test9.jpg', results[0]['rois'])
    r = results[0]
    test_obj = np.unique(r['class_ids'])
    print(test_obj)
    n = len(test_obj)
    for j in range (n):
        x = test_obj[j]
        if x > 0:
            table3[0,x]=1

#    arr = np.reshape(test_obj, (1,n))
#    for j in range (n):
#        table3[0,j] = arr[0,j-1]
#    pd.DataFrame(table3).to_csv("onehot_Test_obj.csv")
    trainX = np.expand_dims(trainX, axis=2)
    test_obj = table3
```

```
    test_obj = np.expand_dims(test_obj, axis=2)

# repeat experiment

    for n in range(repeats):
        y = prediction_model(test_obj)
        print(y)
        room_index = np.argmax(y, axis=1)
        print(room_index)
        if room_index == 0:
            print('bathroom')
        elif room_index == 1:
            print('bedroom')
        elif room_index == 2:
            print('diningroom')
        elif room_index == 3:
            print('kitchen')
        elif room_index == 4:
            print('livingroom')
        else:
            print("Unknown room")

run_test()

def run_validation(repeats=1):
    trainX, trainy, testX, testy = load_dataset(trainpath, train_label,
testpath, test_label)

#    arr = np.reshape(test_obj, (1,n))
#    for j in range (n):
#        table3[0,j] = arr[0,j-1]

    trainX = np.expand_dims(trainX, axis=2)
    test_obj = np.expand_dims(testX, axis =2)

   # repeat experiment
    for n in range(repeats):
        y = prediction_model(test_obj)
        print(y)
        room_index = np.argmax(y, axis=1)
        print(room_index)
        if room_index == 0:
            print('bathroom')
        elif room_index == 1:
            print('bedroom')
        elif room_index == 2:
            print('diningroom')
        elif room_index == 3:
            print('kitchen')
        elif room_index == 4:
            print('livingroom')
        else:
            print("Unknown room")
# run the experiment
run_validation()
```

Appendix D – Conjunction of proposed approaches (Mask R-CNN + CNN) with an 1D CapsNet for IAS generic object detection

In this appendix, the conjunction of Mask R-CNN + CNN (Chapter 6) with the 1D CapsNet (Chapter 5) is presented (The pseudo code was developed during the Covid lockdown. Therefore, this has not been practically implemented). This is done to enable an IAS to perform generic scene and object recognition irrespective of object orientation. Indoor home scene recognition is more dependent on the combination of different objects that constitute a particular scene whereas recognizing particular objects relies specifically on the shape of each object and its orientation. Therefore, the Mask R-CNN + CNN is designed to help an IAS to navigate easily in the indoor home areas and identify its current location. When however, an IAS is expected to handle and move objects, then it will require the 1D CapsNet for both recognising and for handling an object. In such cases the orientation of the object must be known to the IAS at that moment.

The conjunction of Mask RCNN + CNN and 1D CapsNet is shown using a flowchart in Figure 6.13. There can be 4 cases for an IAS to consider:

1. No input received;
2. Input of a particular indoor area (room) that the IAS is expected to reach;
3. Input of an object only which the IAS is expected to find and handle;
4. Both indoor home scene and object inputs, in which an IAS is expected to locate the room find the object and handle and/or move the object.

In case of no input received, the IAS will perform no activity or remain in sleep mode. When a person commands the IAS to go to a particular room, the IAS receives the command as a scene input and activates the Mask RCNN + CNN combination. The IAS keeps navigating until it reaches the desired scene. The navigation for finding the required scene will happen over a defined time frame. If the IAS fails to find the required scene within this time frame, then it will navigate back to the place where it received the command or some other designated area. Once the IAS reaches the desired scene, it checks if it has any further input for finding, and/or handling an object. If there is no such input, then the IAS waits for further input.

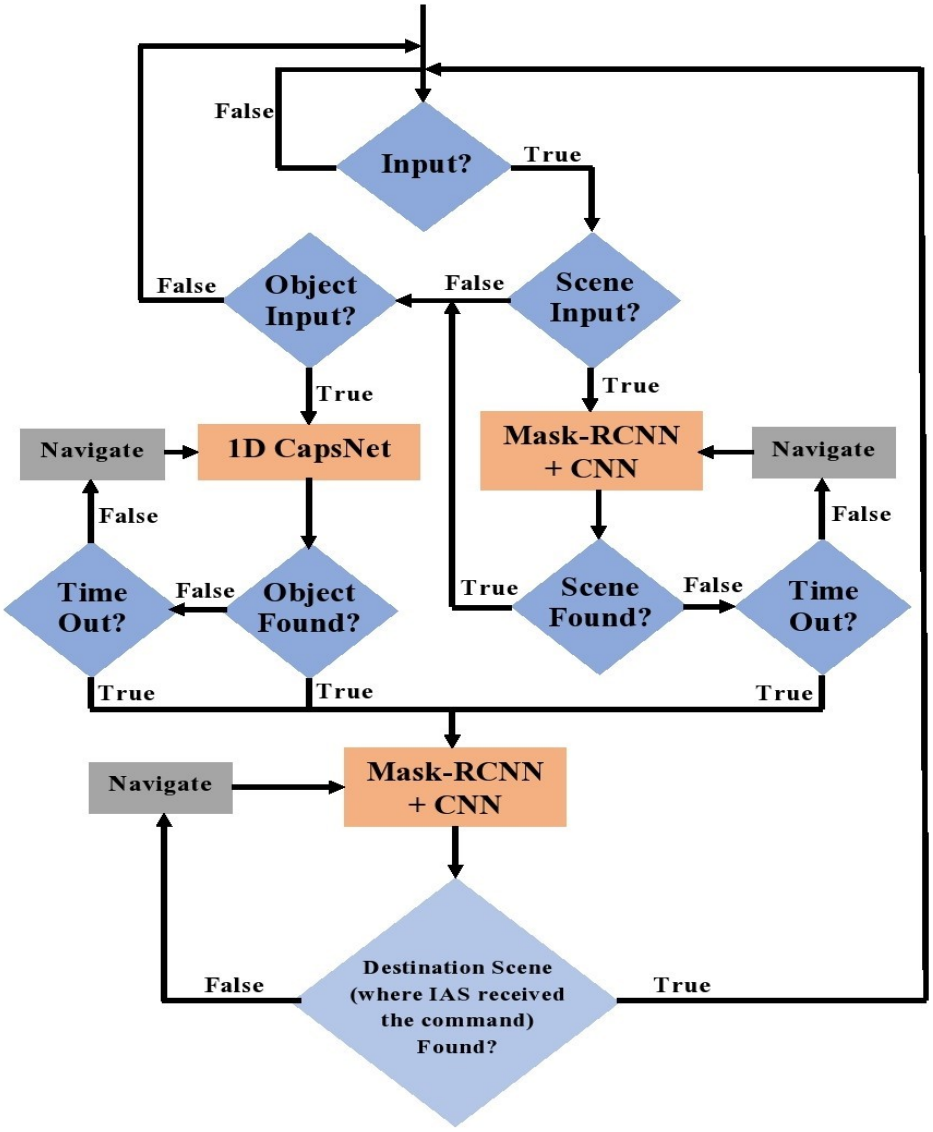


Figure D.1. Flow chart for conjunction of proposed technique (Mask R-CNN + CNN) and 1D CapsNet

Further, if an IAS receives only an object as an input command, then it directly activates the 1D CapsNet module. The IAS will keep trying to find the object by navigating within the vicinity of the area where it happens to be until the desired object is found. Once the object is found, the IAS can, for example, pick up the object and return to the place where it received the command. Navigating back to original starting area the IAS again uses the Mask R-CNN + CNN module to navigate and recognise the scene. Suppose the object is not in the vicinity of the area where IAS is already present. In that case, it searches for the object in other scenes by navigating among different indoor home scenes. There is also a fixed time frame for finding the desired object. If the object is not found within the designated time frame, then the IAS navigates back to the place where it received the input.

In case of both scene an object inputs, the IAS first uses the Mask-RCNN + CNN to find the scene and then uses the 1D CapsNet to find the object. The IAS executes the scene recognition just like the way it executes the task when it receives only scene input. Once, the scene is found within the time frame then the IAS looks for the object input. Once the IAS detects an object input, it executes the object recognition process just like the way it does in the case of only object input. However, in this case, the IAS searches only in the scene that it was first expected to reach.

References

[1] M. E. Pollack, "Intelligent assistive technology: the present and the future," 2007: Springer, pp. 5-6.

[2] M. E. Pollack, "Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment," *AI magazine,* vol. 26, no. 2, p. 9, 2005.

[3] J. Ausubel. "Older people are more likely to live alone in the U.S. than elsewhere in the world." https://www.pewresearch.org/fact-tank/2020/03/10/older-people-are-more-likely-to-live-alone-in-the-u-s-than-elsewhere-in-the-world/ (accessed 6 November, 2021).

[4] N. H. S. United Kingdom. "Loneliness in older people." https://www.nhs.uk/mental-health/feelings-symptoms-behaviours/feelings-and-symptoms/loneliness-in-older-people/ (accessed 6 November, 2021).

[5] D. Dey, K. Nguyen, C. Brockett, and B. Dolan. "HELP! Training assistive indoor agents to ask for assistance via imitation learning - Microsoft Research." https://www.microsoft.com/en-us/research/blog/help-training-assistive-indoor-agents-to-ask-for-assistance-via-imitation-learning/ (accessed 24 November, 2021).

[6] Microsoft. "AI for Accessibility - Microsoft AI." https://www.microsoft.com/en-us/ai/ai-for-accessibility (accessed 24 November, 2021).

[7] D. Leprince-Ringuet. "Facebook wants to help train the robots that will take out your trash and unload your dishwasher." https://www.zdnet.com/article/facebook-wants-to-help-train-the-robots-that-will-take-out-your-trash-and-unload-your-dishwasher/ (accessed 6 November, 2021).

[8] "Matterport and Facebook AI research collaborate to release the world's largest dataset of 3D spaces for academic spaces." https://www.i-micronews.com/matterport-and-facebook-ai-research-collaborate-to-release-the-worlds-largest-dataset-of-3d-spaces-for-academic-research/?cn-reloaded=1 (accessed 6 November, 2021).

[9] A. Quattoni and A. Torralba, "Recognizing indoor scenes," 2009: IEEE, pp. 413-420.

[10] "Important Facts about Falls." Available online: http://www.cdc.gov/HomeandRecreationalSafety/Falls/adultfalls.html. (accessed 05 April, 2021).

[11] "Important Facts about Falls." Available online: http://www.cdc.gov/HomeandRecreationalSafety/Falls/adultfalls.html. (accessed 05/04, 2021).

[12] "Seniors' Falls in Canada: Second Report. Public Health Agency of Canada, 2014. ." Available online: http://www.phac-aspc.gc.ca/seniors-aines/publications/public/injury-blessure/seniors_falls-chutes_ aines/index-eng.php. (accessed 05 April, 2021).

[13] D. Who, "Noncommunicable Disease and Mental Health Cluster, The Injury Chart Book. Fall-related injuries," ed: Geneva, Switzerland: World Health Organization, 2002.

[14] "Chronic Rheumatic Conditions." Available Online: https://www.who.int/chp/topics/rheumatic/en/#:~:text=Worldwide%20estimates%20are%20that%209.6,major%20daily%20activities%20of%20life. (accessed 15 April, 2021).

[15] J. Demirovic *et al.*, "Prevalence of dementia in three ethnic groups: the South Florida program on aging and health," *Annals of epidemiology,* vol. 13, no. 6, pp. 472-478, 2003.

[16] "Alzheimer's Disease Statistics." Available Online: https://alzheimersnewstoday.com/alzheimers-disease statistics/#:~:text=Prevalence,a%20related%20form%20of%20dementia. (accessed 6 April, 2021).
[17] "Diabetes Prevalence." Available Online: https://www.diabetes.co.uk/diabetes-prevalence.html#:~:text=It%20is%20estimated%20that%20415,with%20diabetes%20worldwide%20by%202040. (accessed 02 April, 2021).
[18] "The Future is Elder Care Robots." Available Online: https://waypointrobotics.com/blog/elder-care-robots/. (accessed 09 April, 2021).
[19] C. Mucchiani, P. Cacchione, R. Mead, M. Johnson, and M. Yim, "Preliminary Hardware and System Design Investigation for an Affordable and Mobile Assistive Robot for Elderly Care," 2019.
[20] "Alpha 2, a Humanoid Robot With Social Skills." Available Online: https://spectrum.ieee.org/automaton/robotics/home-robots/ubtech-alpha-2-humanoid-robot. (accessed 11 October, 2018).
[21] "Pillo." Available Online: https://pillohealth.com/. (accessed 21 September, 2018).
[22] "Wakamaru Robot." Available Online: https://robots.ieee.org/robots/wakamaru/. (accessed 15 January, 2021).
[23] "Robot to care for elderly made at University of Salford." Available Online: https://www.bbc.co.uk/news/uk-england-manchester-21590182. (accessed 14 April, 2019).
[24] "Robot man Theo's creation for BBC Christmas." Available Online: https://news-archive.salford.ac.uk/news/articles/2017/robot-man-theos-creation-for-bbc-christmas.html?SQ_DESIGN_NAME=news-portal. (accessed 30 April, 2019).
[25] P. Flandorfer, "Population ageing and socially assistive robots for elderly persons: the importance of sociodemographic factors for user acceptance," *International Journal of Population Research,* vol. 2012, 2012.
[26] A. Haasch *et al.*, "Biron–the bielefeld robot companion," *Dialog,* vol. 11111111, p. 11111111, 2004.
[27] R. Chatila, "Towards cognitive robot companions," 2008: IEEE, pp. 391-391.
[28] "COGNIRON: The Cognitive Robot." Available Online: http://www.cogniron.org/final/Home.php. (accessed 04 March, 2021).
[29] S. Chaudhuri, H. Thompson, and G. Demiris, "Fall detection devices and their use with older adults: a systematic review," *Journal of geriatric physical therapy (2001),* vol. 37, no. 4, p. 178, 2014.
[30] E. J. Porter, "Wearing and Using Personal Emergency," *Journal of Gerontological Nursing,* vol. 31, no. 10, pp. 26-33, 2005.
[31] R. Yared and B. Abdulrazak, "Ambient technology to assist elderly people in indoor risks," *Computers,* vol. 5, no. 4, p. 22, 2016.
[32] J. Fleming and C. Brayne, "Inability to get up after falling, subsequent time on floor, and summoning help: prospective cohort study in people over 90," *Bmj,* vol. 337, 2008.
[33] X. Yu, "Approaches and principles of fall detection for elderly and patient," 2008: IEEE, pp. 42-47.
[34] M. A. Habib, M. S. Mohktar, S. B. Kamaruzzaman, K. S. Lim, T. M. Pin, and F. Ibrahim, "Smartphone-based solutions for fall detection and prevention: challenges and open issues," *Sensors,* vol. 14, no. 4, pp. 7181-7208, 2014.
[35] F. Sposaro and G. Tyson, "iFall: an Android application for fall monitoring and response," 2009: IEEE, pp. 6119-6122.

[36] I. C. Lopes, B. Vaidya, and J. J. P. C. Rodrigues, "Sensorfall-an accelerometer based mobile application," 2009: IEEE, pp. 1-6.
[37] J. T. Perry, S. Kellog, S. M. Vaidya, J.-H. Youn, H. Ali, and H. Sharif, "Survey and evaluation of real-time fall detection approaches," 2009: IEEE, pp. 158-164.
[38] A. K. Bourke, K. J. O'Donovan, and G. M. ÓLaighin, "Distinguishing falls from normal ADL using vertical velocity profiles," 2007: IEEE, pp. 3176-3179.
[39] A. K. Bourke, K. J. O'Donovan, and G. Olaighin, "The identification of vertical velocity profiles using an inertial sensor to investigate pre-impact detection of falls," *Medical Engineering & Physics,* vol. 30, no. 7, pp. 937-946, 2008.
[40] T. Zhang, J. Wang, L. Xu, and P. Liu, "Fall detection by wearable sensor and one-class SVM algorithm," in *Intelligent computing in signal processing and pattern recognition*: Springer, 2006, pp. 858-863.
[41] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford, "A hybrid discriminative/generative approach for modeling human activities," 2005, vol. 5: Citeseer, 2005 ed.
[42] L. Tong, Q. Song, Y. Ge, and M. Liu, "HMM-based human fall detection and prediction method using tri-axial accelerometer," *IEEE Sensors Journal,* vol. 13, no. 5, pp. 1849-1856, 2013.
[43] B. Florentino-Liano, N. O'Mahony, and A. Artés-Rodríguez, "Hierarchical Dynamic Model for Human Daily Activity Recognition," 2012, pp. 61-68.
[44] J. Yin, Q. Yang, and J. J. Pan, "Sensor-based abnormal human-activity detection," *IEEE Transactions on Knowledge and Data Engineering,* vol. 20, no. 8, pp. 1082-1090, 2008.
[45] M. Yu, S. M. Naqvi, A. Rhuma, and J. Chambers, "One class boundary method classifiers for application in a video-based fall detection system," *IET computer vision,* vol. 6, no. 2, pp. 90-100, 2012.
[46] Y. Wang, K. Wu, and L. M. Ni, "Wifall: Device-free fall detection by wireless networks," *IEEE Transactions on Mobile Computing,* vol. 16, no. 2, pp. 581-594, 2016.
[47] D. Zhang, H. Wang, Y. Wang, and J. Ma, "Anti-fall: A non-intrusive and real-time fall detector leveraging CSI from commodity WiFi devices," 2015: Springer, pp. 181-193.
[48] M. S. Khan, M. Yu, P. Feng, L. Wang, and J. Chambers, "An unsupervised acoustic fall detection system using source separation for sound interference suppression," *Signal processing,* vol. 110, pp. 199-210, 2015.
[49] G. I. Parisi and S. Wermter, "Hierarchical SOM-based detection of novel behavior for 3D human tracking," 2013: IEEE, pp. 1-8.
[50] R. Barber, J. Crespo, C. Gómez, A. C. Hernámdez, and M. Galli, "Mobile robot navigation in indoor environments: Geometric, topological, and semantic navigation," in *Applications of Mobile Robots*: IntechOpen, 2018.
[51] "The Index Project." Available Online: https://theindexproject.org/post/nursebot. (accessed 14 April, 2021).
[52] B. Ando, S. Baglio, and C. O. Lombardo, "RESIMA: An assistive paradigm to support weak people in indoor environments," *IEEE Transactions on Instrumentation and Measurement,* vol. 63, no. 11, pp. 2522-2528, 2014.
[53] Y.-J. Chang and T.-Y. Wang, "Indoor wayfinding based on wireless sensor networks for individuals with multiple special needs," *Cybernetics and Systems: An International Journal,* vol. 41, no. 4, pp. 317-333, 2010.
[54] A. L. Liu *et al.*, "Indoor wayfinding: Developing a functional interface for individuals with cognitive impairments," *Disability and Rehabilitation: Assistive Technology,* vol. 3, no. 1-2, pp. 69-81, 2008.

[55] B. Li, J. P. Munoz, X. Rong, J. Xiao, Y. Tian, and A. Arditi, "ISANA: wearable context-aware indoor assistive navigation with obstacle avoidance for the blind," 2016: Springer, pp. 448-462.
[56] Y.-J. Chang, C.-N. Chen, L.-D. Chou, and T.-Y. Wang, "A novel indoor wayfinding system based on passive RFID for individuals with cognitive impairments," 2008: IEEE, pp. 108-111.
[57] H. A. Yanco, "Wheelesley: A robotic wheelchair system: Indoor navigation and user interface," in *Assistive technology and artificial intelligence*: Springer, 1998, pp. 256-268.
[58] A. O. Caffo *et al.*, "Comparing two different orientation strategies for promoting indoor traveling in people with Alzheimer's disease," *Research in developmental disabilities,* vol. 35, no. 2, pp. 572-580, 2014.
[59] M. Kahraman and C. Turhan, "An intelligent indoor guidance and navigation system for the visually impaired," *Assistive Technology,* pp. 1-9, 2021.
[60] G. Ballestin and T. Zielinska, "Indoor robot navigation and mapping using sensory fusion," in *Mechanism and Machine Science*: Springer, 2021, pp. 279-292.
[61] W. Yan, C. Weber, and S. Wermter, "Learning indoor robot navigation using visual and sensorimotor map information," *Frontiers in neurorobotics,* vol. 7, p. 15, 2013.
[62] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv preprint arXiv:2005.13857,* 2020.
[63] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems,* vol. 30, no. 11, pp. 3212-3232, 2019.
[64] L. Xie, F. Lee, L. Liu, K. Kotani, and Q. Chen, "Scene recognition: A comprehensive survey," *Pattern Recognition,* vol. 102, p. 107205, 2020.
[65] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85-117, 2015.
[66] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature,* vol. 521, no. 7553, p. 436, 2015.
[67] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," 2010: IEEE, pp. 253-256.
[68] K. Hara, D. Saito, and H. Shouno, "Analysis of function of rectified linear unit used in deep learning," 2015: IEEE, pp. 1-8.
[69] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synthesis Lectures on Computer Vision,* vol. 8, no. 1, pp. 1-207, 2018.
[70] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.
[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012, pp. 1097-1105.
[72] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360,* 2016.
[73] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2014: Springer, pp. 818-833.
[74] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.
[75] C. Szegedy *et al.*, "Going deeper with convolutions," 2015, pp. 1-9.

[76] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence,* vol. 38, no. 1, pp. 142-158, 2016.
[77] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015, pp. 1026-1034.
[78] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," 2010, pp. 249-256.
[79] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016, pp. 770-778.
[80] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2017, vol. 4, p. 12.
[81] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2016, pp. 2818-2826.
[82] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016: Springer, pp. 630-645.
[83] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400,* 2013.
[84] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2015, pp. 3431-3440.
[85] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature Pyramid Networks for Object Detection," 2017, vol. 1, 2 ed., p. 3.
[86] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015: Springer, pp. 234-241.
[87] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," 2016: IEEE, pp. 565-571.
[88] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *arXiv preprint arXiv:1511.00561,* 2015.
[89] A. Chaurasia and E. Culurciello, "Linknet: Exploiting encoder representations for efficient semantic segmentation," 2017: IEEE, pp. 1-4.
[90] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," 2017: IEEE, pp. 1175-1183.
[91] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," 2017, pp. 2881-2890.
[92] G. Lin, A. Milan, C. Shen, and I. D. Reid, "RefineNet: Multi-path Refinement Networks for High-Resolution Semantic Segmentation," 2017, vol. 1, 2 ed., p. 5.
[93] M. A. Islam, M. Rochan, N. D. B. Bruce, and Y. Wang, "Gated feedback refinement network for dense image labeling," 2017: IEEE, pp. 4877-4885.
[94] S. Hong, H. Noh, and B. Han, "Decoupled deep neural network for semi-supervised semantic segmentation," 2015, pp. 1495-1503.
[95] N. Souly, C. Spampinato, and M. Shah, "Semi and weakly supervised semantic segmentation using generative adversarial network," *arXiv preprint arXiv:1703.09695,* 2017.
[96] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," 2016, pp. 379-387.
[97] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015 2015, pp. 1440-1448.
[98] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence,* vol. 37, no. 9, pp. 1904-1916, 2015.

[99] M.-R. Hsieh, Y.-L. Lin, and W. H. Hsu, "Drone-based object counting by spatially regularized regional proposal network," 2017, vol. 1.
[100] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis & Machine Intelligence,* no. 6, pp. 1137-1149, 2017.
[101] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge 2007 (voc 2007) results (2007)," ed, 2008.
[102] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," 2014: Springer, pp. 740-755.
[103] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017: IEEE, pp. 2980-2988.
[104] 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, pp. 1-5.
[105] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE transactions on pattern analysis and machine intelligence,* 2017.
[106] L. Wang, S. Guo, W. Huang, Y. Xiong, and Y. Qiao, "Knowledge guided disambiguation for large-scale scene classification with multi-resolution CNNs," *IEEE Transactions on Image Processing,* vol. 26, no. 4, pp. 2055-2068, 2017.
[107] P. Espinace, T. Kollar, N. Roy, and A. Soto, "Indoor scene recognition by a mobile robot through adaptive object detection," *Robotics and Autonomous Systems,* vol. 61, no. 9, pp. 932-947, 2013.
[108] P. Espinace, T. Kollar, A. Soto, and N. Roy, "Indoor scene recognition through object detection," 2010: IEEE, pp. 1406-1413.
[109] A. Shapiro, "Monte Carlo sampling methods," *Handbooks in operations research and management science,* vol. 10, pp. 353-425, 2003.
[110] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016, pp. 779-788.
[111] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *arXiv preprint,* vol. 1612, 2016.
[112] P. Wu, Y. n. Li, F. Yang, L. Kong, and Z. Hou, "A CLM-based method of indoor affordance areas classification for service robots," *Jiqiren/Robot,* vol. 40, no. 2, pp. 188-194, 2018.
[113] E. B. Sudderth, A. Torralba, W. T. Freeman, and A. S. Willsky, "Learning hierarchical models of scenes, objects, and parts," 2005, vol. 2: IEEE, pp. 1331-1338.
[114] N. Ali and B. Zafar, "15-scene image dataset," *Figshare,* 2018.
[115] M. Afif, R. Ayachi, Y. Said, and M. Atri, "Deep Learning Based Application for Indoor Scene Recognition," *Neural Processing Letters,* pp. 1-11, 2020.
[116] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," 2018, pp. 5010-5019.
[117] Z. Wu *et al.*, "3d shapenets: A deep representation for volumetric shapes," 2015, pp. 1912-1920.
[118] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," 2015, pp. 945-953.
[119] 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: IEEE, pp. 1578-1584.

[120] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017, pp. 652-660.
[121] L. Noriega, "Multilayer perceptron tutorial," *School of Computing. Staffordshire University,* 2005.
[122] J. Li, B. M. Chen, and G. Hee Lee, "So-net: Self-organizing network for point cloud analysis," 2018, pp. 9397-9406.
[123] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," *arXiv preprint arXiv:1707.02392,* 2017.
[124] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413,* 2017.
[125] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," 2019, pp. 9621-9630.
[126] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on χ-transformed points," 2018, pp. 828-838.
[127] J. Jiang, D. Bao, Z. Chen, X. Zhao, and Y. Gao, "MLVCNN: Multi-Loop-View Convolutional Neural Network for 3D Shape Retrieval," 2019, vol. 33, pp. 8513-8520.
[128] K. Zhang, M. Hao, J. Wang, C. W. de Silva, and C. Fu, "Linked dynamic graph CNN: Learning on point cloud via linking hierarchical features," *arXiv preprint arXiv:1904.10014,* 2019.
[129] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," 2017, pp. 3856-3866.
[130] R. Lambert, "Investigation: Capsule Nets for Content-Based 3D Model Retrieval," *Dimension,* vol. 30, p. 30x30x30.
[131] A. Ahmad, "Object Recognition in 3D data using Capsules," 2018.
[132] A. Cheraghian and L. Petersson, "3dcapsule: Extending the capsule architecture to classify 3d point clouds," 2019: IEEE, pp. 1194-1202.
[133] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," 2011: Springer, pp. 44-51.
[134] M. K. Patrick, A. F. Adekoya, A. A. Mighty, and B. Y. Edward, "Capsule networks–a survey," *Journal of King Saud University-Computer and Information Sciences,* 2019.
[135] A. Basu, L. Petropoulakis, G. Di Caterina, and J. Soraghan, "Indoor home scene recognition using capsule neural networks," *Procedia Computer Science,* vol. 167, pp. 440-448, 2020.
[136] A. Basu, K. Kaewrak, L. Petropoulakis, G. Di Caterina, and J. J. Soraghan, "Modified Capsule Neural Network (Mod-CapsNet) for indoor home scene recognition," 2020.
[137] X. Song, S. Jiang, B. Wang, C. Chen, and G. Chen, "Image representations with spatial object-to-object relations for RGB-D scene recognition," *IEEE Transactions on Image Processing,* vol. 29, pp. 525-537, 2019.
[138] X. Song, C. Chen, and S. Jiang, "RGB-D scene recognition with object-to-object relation," 2017, pp. 600-608.
[139] J. West, D. Ventura, and S. Warnick, "Spring research presentation: A theoretical foundation for inductive transfer," *Brigham Young University, College of Physical and Mathematical Sciences,* vol. 1, no. 08, 2007.
[140] G. Patterson and J. Hays, "Sun attribute database: Discovering, annotating, and recognizing scene attributes," 2012: IEEE, pp. 2751-2758.
[141] Amlan Basu, Lykourgos Petropoulakis, Gaetano Di Caterina, and John Soraghan. "Assistive technology evolving as intelligent system." *International Conference On Computational Vision and Bio Inspired Computing*, pp. 289-303, 2018.

[142] Minh Tanh, Viet-Khoa Vo-Ho, Kyle Quinn, Hien Nguyen, Khoa Luu, and Ngan Le. "CapsNet for Medical Image Segmentation." *arXiv preprint arXiv:2203.08948* (2022).
[143] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." *IEEE conference on computer vision and pattern recognition*, pp. 248-255, 2009.
[144] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang et al. "Imagenet large scale visual recognition challenge." *International journal of computer vision*, pp.211-252, 2015.
[145] Mark Everingham, S. M. Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes challenge: A retrospective." *International journal of computer vision*, pp. 98-136, 2015.
[146] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. "Semantic understanding of scenes through the ade20k dataset." *International Journal of Computer Vision* 127, no. 3 (2019): 302-321.
[147] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. "Sun database: Large-scale scene recognition from abbey to zoo." In *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 3485-3492, 2010.
[148] Xiao, Jianxiong, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. "Sun database: Exploring a large collection of scene categories." *International Journal of Computer Vision*, pp. 3-22, 2016.
[149] Genevieve Patterson, Chen Xu, Hang Su, and James Hays. "The sun attribute database: Beyond categories for deeper scene understanding." *International Journal of Computer Vision*, pp. 59-81, 2014.
[150] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. "Sun rgb-d: A rgb-d scene understanding benchmark suite." *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 567-576. 2015.
[151] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. "Learning deep features for scene recognition using places database." *Advances in neural information processing systems*, 2014.
[152] Nouman Ali, Bushra Zafar, 15-Scene Image Dataset. figshare Dataset, 2018 https://doi.org/10.6084/m9.figshare.7007177.v1
[153] Li-Jia Li, and Li Fei-Fei. "What, where and who? classifying events by scene and object recognition." *11th international conference on computer vision*, pp. 1-8, 2007.
[154] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. "Segmentation and recognition using structure from motion point clouds." *European conference on computer vision*, pp. 44-57. 2008.
[155] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. "Semantic object classes in video: A high-definition ground truth database." *Pattern Recognition Letters* 30, pp. 88-97, 2009.
[156] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The cityscapes dataset for semantic urban scene understanding." *IEEE conference on computer vision and pattern recognition*, pp. 3213-3223. 2016.
[157] Li Fei-Fei, Rob Fergus, and Pietro Perona. "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories." *IEEE conference on computer vision and pattern recognition workshop*, pp. 178-178, 2004.
[158] Gregory Griffin, Alex Holub, and Pietro Perona. "Caltech-256 object category dataset." 2007.

[159] Gonzalez, Rafael C. *Digital image processing*. Pearson education india, 2009.
[160] Giovanni Lucca França da Silva, Thales Levi Azevedo Valente, Aristófanes Corrêa Silva, Anselmo Cardoso de Paiva, and Marcelo Gattass. "Convolutional neural network-based PSO for lung nodule false positive reduction on CT images." *Computer methods and programs in biomedicine*, pp. 109-118, 2018.
[161] Donald F. Specht, "A general regression neural network." *IEEE transactions on neural networks*, pp. 568-576, 1991.
[162] Abel Brown, "Introduction to object detection and image segmentation," https://on-demand.gputechconf.com/gtcdc/2017/presentation/dc7217-abel-brown-deep-learning-object-detection-and-segmentation.pdf (accessed: 11 November, 2021)