university of groningen

faculty of science and engineering

MASTER'S THESIS
ARTIFICIAL INTELLIGENCE

3D_DEN: Open-ended 3D Object Recognition Using Dynamically Expandable Networks

Author
SUDHAKARAN JAIN
S3558487

Primary Supervisor: Dr. Hamidreza Kasaei
Secondary Supervisor: Prof. Dr. Herbert Jaeger

Artificial Intelligence, University of Groningen

November 20, 2020

Contents

List of Figures **3**

List of Tables **5**

1 Introduction **8**
1.1 Background 8
1.2 Research Questions 9
1.2.1 Which 3D_DEN variant yields the best results when integrated with different pre-trained networks? Also, does it outperform the current state-of-the-art? 9
1.2.2 How does the best performing 3D_DEN variant compare with offline-trained model? 10
1.2.3 How does the best performing 3D_DEN variant perform in real-world scenarios? 10
1.3 Outline 10

2 Theoretical Background **11**
2.1 Convolutional Neural Networks (CNNs) 11
2.1.1 Convolution [1] 11
2.1.2 Pooling [1] 13
2.1.3 Activation Layer 13
2.1.4 Fully-Connected (FC) Layers 15
2.2 Transfer Learning 16
2.2.1 Pre-training 16
2.3 Literature Review 17
2.3.1 Continual Learning (CL) Approaches 17
2.3.2 Object Recognition Approaches 19
2.4 Comparing 3D_DEN with other approaches 21

3 Methodology **23**
3.1 Data Sampling (Pure Rehearsal) 24
3.2 Selective Retraining 25
3.3 Dynamic Expansion 26
3.4 Dataset and Pre-processing 28

4 Experiment and Results **29**
4.1 Open-Ended Training Evaluation 29
4.2 Offline Evaluation using Grid Search 33
4.3 Evaluation on Real-Time Robot 35

5 Discussion and Conclusion **39**
5.1 Discussion 39
5.1.1 Constant increase in DEN size 39
5.1.2 Inefficiency of Selective Retraining 39
5.1.3 Inability to learn color features 40
5.1.4 Memory requirements 40
5.1.5 Answers to research questions 40
5.2 Conclusion 41
5.2.1 Future Work 41

Bibliography **43**

List of Figures

2.1 A CNN architecture for recognition of digits as shown in article [2]. . . . 12
2.2 Convolution operations in a CNN as shown in [3]. Here, the single-channel (d=1) input image is of dimension 3×4 and filter is 2×2 with stride of 1. Each letter represents a pixel value in both these entities. When the filter is slid over the image, dot products are computed correspondingly (6 times in this case) which form the output. 14
2.3 The image is taken from [1]. In the left section, pooling operation is done on the input volume of size $224 \times 224 \times 64$ to get an output volume $112 \times 112 \times 64$. Note that the volume is not reduced along the depth dimension (or channels). In the right section, Max-pooling on an activation map is shown. Each output is obtained by taking maximum value across 2×2 windows. 15
2.4 This image was taken from the paper [4]. It shows a Venn diagram illustrating recent popular papers on Continual Learning: PNN [5], CWR [6], EWC [7], SI [8], GEM [9], LWF [10], ExStream [11], FearNet [12], ICARL [13], MeRGAN [14], GDM [15], AR1 [16], Pure Rehearsal and GR [17]. Rehearsal and generative replay are subsets of replay strategies [4]. 17

3.1 This is the proposed architecture of our `3D_DEN` model. Initially, three representative views are chosen from a set of multi-view images for a given 3D object. Then, each of them is converted to a single channel (grey-scale) image and later merged to form a 3-channel image. Now, this image is fed to a pre-trained network, and the extracted features are flattened. Finally, we attach two DEN layers to the model which give the output. 23
3.2 Selective Retraining: First, the weights between top-most hidden layer and output layer are trained. In next step, we select nodes in the top-most hidden layer that have non-zero values to the new output node. Then, we perform a layer-wise search to identify rest of the nodes in sparse bottom layers, that have non-zero connections to these selected nodes. Selected sub-network is represented by red color. 26
3.3 Dynamic Expansion: New nodes are added to DEN layers and trained, while rest of the nodes are kept constant. In the end, new nodes that were found useless are removed. Here, newly added nodes are represented by red color. 27

4.1 Summary of open-ended evaluation: (*top*) shows the timeline of GCA for all three models; (*middle*) shows training time (approx) needed by our models while learning new tasks; (*bottom*) shows the increment in number of neurons in DEN layers for all models. 32
4.2 Results of offline evaluation using grid search is shown above. The best parametric configuration is represented by green-colored connection. Whereas, the red and purple-colored connections show the parametric configurations which achieved second best and third best results, respectively. 34
4.3 Our experimental setup for real-robot experiment consists of a Kinect sensor to perceive the environment, and a UR5e robot to act upon the environment. Throughout the `serve_a_drink` experiment, we use four instances of three object categories including *bottle*, *cup*, and *plant*. It should be noted that similar instances to the selected objects exist in the ModelNet40 dataset. 35
4.4 Image taken from [18]. Each depicted module in the diagram is run as a ROS [19] service and the connections show the flow of information between them [18]. The highlighted red module is replaced by our `3D_DEN`. . 36
4.5 Comparison between a real-world object with the instances used for training: (a) shows the partial point cloud of the vase object captured by the Kinect sensor in our experiment; (b) and (c) show a vase and plant instance from the ModelNet40 dataset, on which our model was trained. . . 37
4.6 A sequence of snapshots from the real-robot evaluation: (a) This snapshot shows our model `3D_DEN` being used to recognize the objects present on the table; (b) The robot grasps the bottle after recognizing it; (c) The robot manipulates the bottle on top of the cup and tries to pour a drink from the bottle into the cup. In these images, the pose of objects is shown by green 3D bounding boxes, and the recognition results are displayed on top of the objects in red. 38

List of Tables

4.1 Properties of Feature Extractor Networks used in Experiment 29
4.2 Result of Open-Ended Evaluation on ModelNet40 Dataset 31
4.3 Grid Search Parameters for Offline Evaluation 34

Abstract

Service robots, in general, have to work independently and adapt to the dynamic changes in the environment. One important aspect in such scenarios is to continually learn to recognize newer object categories when they become available. This combines two main research problems namely continual learning and 3D object recognition. Most of the existing research approaches include the use of deep Convolutional Neural Networks (CNNs) focusing on image datasets. A modified approach might be needed for continually learning 3D object categories. A major concern in using CNNs is the problem of catastrophic forgetting when a model tries to learn a new task. Despite various recent proposed solutions to mitigate this problem, there still exist a few side-effects (such as time/computational complexity) of such solutions. We propose a model capable of learning 3D object categories in an open-ended fashion by employing a deep transfer learning-based approach combined with dynamically expandable layers, which also makes sure that these side-effects are minimized to a great extent. We show that this model sets a new state-of-the-art standard with regards to accuracy and also substantially minimizes computational complexity.

Acknowledgements

I would like to express my sincere appreciation to my primary internal supervisor, Dr. Hamidreza Kasaei for his guidance and support throughout the project. Especially, the feedback and ideas provided by him during our meetings were truly insightful. I also would like to pay my special regards to my secondary supervisor, Prof.Dr.Herbert Jaeger for devoting his valuable time to this project and also for providing timely feedback. A sincere thanks also go to my friend, Mr.Mohit Deorukhkar for helping me to edit diagrams and video needed for my thesis.
Finally, I would like to thank the Center for Information Technology of the University of Groningen for giving me access to the Peregrine, through which I was able to implement my thesis work. Their timely responses to my queries were also very helpful.

Chapter 1

Introduction

1.1 Background

Service robots are generally used in domestic environments where they have to work independently on some given task. Such environments can be changing frequently or rarely. Robots should be able to adapt to these changes and efficiently manipulate objects to achieve some specific goal assigned to it. Or in general, an ideal robot should intelligently perceive the events occurring in its surroundings and act accordingly. One of the main concerns in such scenarios is learning about new object categories in an open-ended fashion using a few training instances of the object categories. Such a learning process is also called open-ended/continual or lifelong learning [20]. It poses a big challenge in the field of Artificial General Intelligence (AGI). Hence, it can be seen as a vital ability required for a robot to perform its day-to-day work.

Even though a lot of attention has been given towards continual learning scenarios, researchers have mainly focused to test their proposed model on 2D image datasets, usually MNIST [21]. On the other hand, very little attention has been given to the problem of continual learning of 3D objects. Despite the presence of various papers to improve 3D object classification, open-ended recognition of 3D objects still poses a problem that lacks an efficient solution. Therefore, it has a lot of research scope for improvement.

In recent times, the research community has been giving much attention to deep Convolutional Neural Networks (CNNs) for continual learning tasks on image datasets. When the output object categories are pre-determined (or fixed) and a large number of training instances for each category is available that also resemble the test set, CNNs can give satisfactory results [22]. CNNs can be considered incremental in the sense that we can sequentially train each task in such a way to improve the model's knowledge about its previously learned tasks. That is, in incremental learning, given the predefined set of output categories, representation of every category is improved while learning each new task. For example, let us consider a CNN-based model for dog breed classification with 10 categories. Here, we can incrementally re-train the model by giving a few instances from every category at each training stage (here, considered a task). This makes the model to

improve its existing knowledge about every breed category with each subsequent task.

On the contrary, CNNs do not readily support open-ended learning, as this demands new tasks to be learned which may or may not be related to previously known tasks, and thus making the above training approach non-applicable. To be specific, in some open-ended scenarios, the model needs to learn newer categories (here, considered a task) using a few training examples presented over a period of time. This mainly leads to overwriting the network's learned weights every time a new task (defined with new output node) comes into play. This indeed, causes Catastrophic Forgetting [23], a major limitation of CNNs. The problem can be defined as forgetting the previous tasks while the model is being trained further on newer tasks [23]. It usually happens due to overwriting the values of weights learned from previous tasks, upon the start of training on newer tasks. In addition to the problem above, learning to recognize 3D objects adds more complications to the whole issue.

We try to overcome the above-mentioned issues by proposing a novel deep learning dynamic architectural model, and methods to train it that improves open-ended 3D object recognition. To be specific, we aim to find whether we can build a model with dynamically expandable architecture that is capable of learning a substantially large number of 3D object categories when they become available, without catastrophically forgetting already known ones. Our proposed model is called `3D_DEN` where **DEN** stands for 'Dynamically Expandable Networks', named after one of our major parent paper [24] that inspired our work. Figure 3.1 shows how our model `3D_DEN` looks like.

1.2 Research Questions

To accomplish the aim of our thesis, we will conduct various experiments on *Princeton ModelNet40* which is a 3D object dataset. The specific questions we seek to answer are as follows:

1.2.1 Which `3D_DEN` variant yields the best results when integrated with different pre-trained networks? Also, does it outperform the current state-of-the-art?

Here, we experiment with different pre-trained networks to find the best one for our `3D_DEN` model and which might outperform the current state-of-the-art [22] for open-ended 3D object recognition. We will try to find the best `3D_DEN` variant not only with regards to accuracy but also in terms of computational cost. In the end, the best `3D_DEN` variant will be compared to current state-of-the-art, which is OrthographicNet [22].

1.2.2 How does the best performing `3D_DEN` variant compare with offline-trained model?

We conduct a comparative experiment between open-ended evaluation and offline evaluation to check how our model performs with an offline trained model.

To elaborate, we will build different offline-trained models each with a unique parametric configuration (using different sizes of Fully-Connected (FC) layers and optimizers) that is governed by a grid-search, to eventually find the best performing one. Later, this best offline-trained model is compared with the best `3D_DEN` variant. Mainly, we seek to compare the size of our DEN layers (discussed in Chapter 3) with fixed size FC-layers of the offline-trained model. Secondly, the type of optimization used by both of them will also be compared.

1.2.3 How does the best performing `3D_DEN` variant perform in real-world scenarios?

In the end, we conduct a robotic experiment, where the task given to the robot is to serve a drink. Here, the robot uses our `3D_DEN` model to recognize real-world objects on a table so that it can manipulate them to execute the `serve_a_drink` scenario. This experiment is conducted to confirm whether our proposed model performs well in real-world situations, especially when integrated into robots.

1.3 Outline

The remainder of this report is organized in the following way. Chapter 2 explains the theoretical background of our work including the reviews of recent work done in continual learning and 3D object recognition. Next, the detailed methodologies of our proposed model namely `3D_DEN` are explained in Chapter 3. Chapter 4 is about the experimental setup and results where we explain in detail the performance of our model compared to the existing state-of-the-art. This is followed by Chapter 5 where we discuss some flaws in our approach and conclude with final remarks on our direction of future work.

Chapter 2

Theoretical Background

In this chapter, we start by giving a brief background about CNNs introduced by LeCunn et al. [25]. They are currently considered the best deep neural network models for image recognition and its related problems. We then comprehend another important concept called *transfer learning*, which describes how we can use these CNNs as pre-trained networks while building a model architecture. Further, we review the latest research papers which made use of CNNs-based models for open-ended object recognition problems. Finally, we compare how our proposed model will be differing from them.

2.1 Convolutional Neural Networks (CNNs)

A CNN consists of a series of different layers (usually convolution followed by Fully-Connected (FC) layers) that takes a two-dimensional input and produces the output. Generally, this two-dimensional data is an image, while the output is a sequence of probability scores of possible labels. Figure 2.1 shows a CNN classifier for digits, from the article [2]. CNNs consist of various functions that are performed in different layers as discussed below.

2.1.1 Convolution [1]

The first and most important component of a CNN is the convolution block (or layer) which is responsible for extracting features through convolution operation. Depending upon the problem at hand, we can choose to have one or more of these blocks for efficient feature extraction. Each of these blocks consists of many learnable parameters called filters. A filter is a set of 2-D matrices stacked together (which makes it 3-dimensional) that operates on a small area of the input, with their height and width defined by the user (generally much less than input's height and width). The depth of the filter is kept the same as the depth of input, i.e., the number of channels.

During the forward pass, the filter is made to slide over the image across both width and height dimensions to compute the dot product between the values in the filter and the image pixels present in this sliding window area. The values from the dot products are

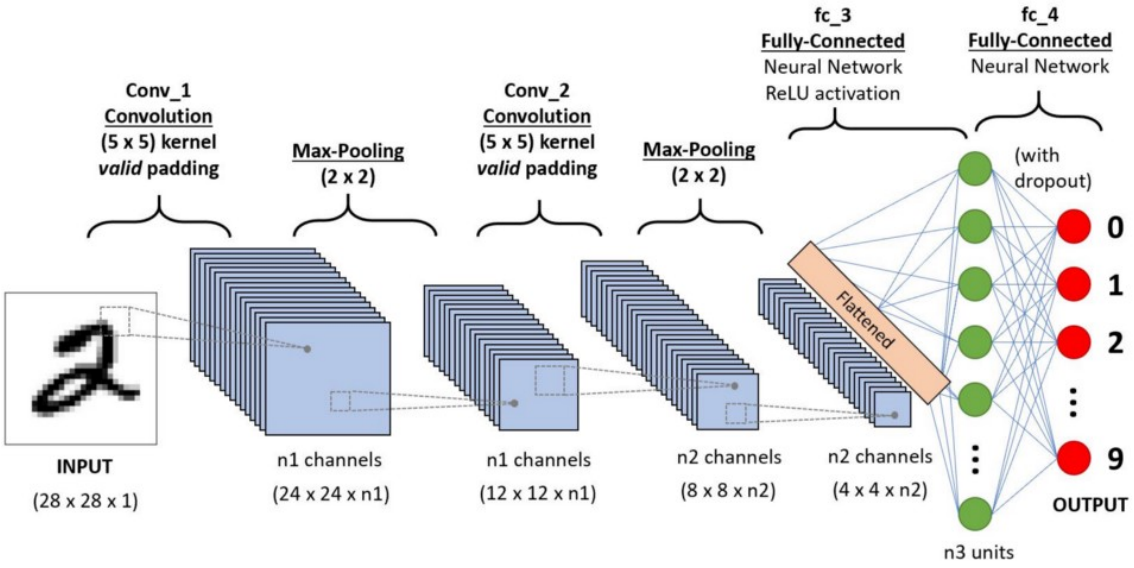


Figure 2.1: A CNN architecture for recognition of digits as shown in article [2].

summed across the depth dimension and added with a bias. Every such computed value is indeed passed to a non-linear activation function to finally obtain a 2-dimensional output grid called an activation map. It should be noted that each sliding operation computes one pixel of this activation map. The below equation explains this convolution operation (output after applying the activation function) performed on an input image.

$$O_k = \xi\left(\left(\sum_{i}^{d}\sum_{j}^{s=3\times3} f_{i,j} \times x_{i,j}\right) + b_f\right)$$

where, O_k is the output value of pixel at position k; ξ denotes a non-linear activation function; d is the depth (no. of channels) of image; s is the shape of filter with respect to width and height dimensions (generally, 3×3); f represents the filter; x is the input image and b_f represents the bias for the filter f. The working of convolution operation on a 2-D image is shown in Figure 2.2, referred from [3].

During training (backward pass), the network learns the filter parameters that get activated when they recognize certain features present at any arbitrary place in the input. A stack of these filters is placed along the depth dimension in each block which thereby results in a stack of 2-D activation maps that constitute the output volume. CNNs gained much popularity due to two major advantages, as discussed in [1]. We briefly explain them as follows.

Sparse Connections [1]

In a neural network model where each node in a layer is connected to all nodes in the previous layer, computing an output involves several matrix multiplications. Every connection

in between layers is given a separate weight parameter that is learned during training. When dealing with multi-dimensional inputs like images, the computational cost when training on such type of network model becomes very high. The key features present in the images such as edges occupy only a small area. To extract such features, there is no need to fully connect all the layers. This is the reason behind the usage of filters in convolution layers of CNNs. Usually, these learnable filters have their dimensions much smaller than the images. During an image classification task, when these filters are slid over the image, each filter tries to capture some specific features of that image output category. In simple terms, every filter learns a generic distribution which activates it when some specific features are found. Having such sparse connectivity reduces computational overhead to a great extent.

Parameter sharing [1]

When the weight parameters are used more than once in the network, it is called parameter sharing [1]. This property is executed by the filters of CNNs. As the main features such as edges maybe be present anywhere in an image, it is ideal to share the same filter (parameters) all across the image for feature extraction. It should be noted when we say parameter sharing, it means that the same 2-D matrices from all channels (along depth dimension) of a filter will be used to slide over each window (sliced depth-wise) of the image. Therefore, it drastically reduces the total number of parameters required by the model.

2.1.2 Pooling [1]

Usually, a pooling layer is present between two consecutive convolution blocks. This layer reduces the size of each activation map obtained from the previous layer by down-sampling it. This eventually reduces the parameters needed in further blocks. It also helps to prevent over-fitting of the model. Moreover, performing convolution operations using filters followed by pooling operations causes the layers in CNNs to gain translation invariance property. There are different types of pooling operations (such as Max, Average). Though, the most common one is Max pooling. Figure 2.3 shows Max-pooling operation as depicted in [1]. Here, each filter is of size 2×2 having a stride (number of pixels to slide) of 2. Pooling operation then reduces each activation map to one-fourth of its actual size, drastically down-sampling it to keep only prominent features. The above reasons make CNN a widely popular architecture for image classification.

2.1.3 Activation Layer

The output of a convolution operation is first sent to an activation layer before pooling is applied. It is slightly analogous to activation thresholds present between the biological neuron connections. In this section, we will describe some of the most commonly used activation functions.

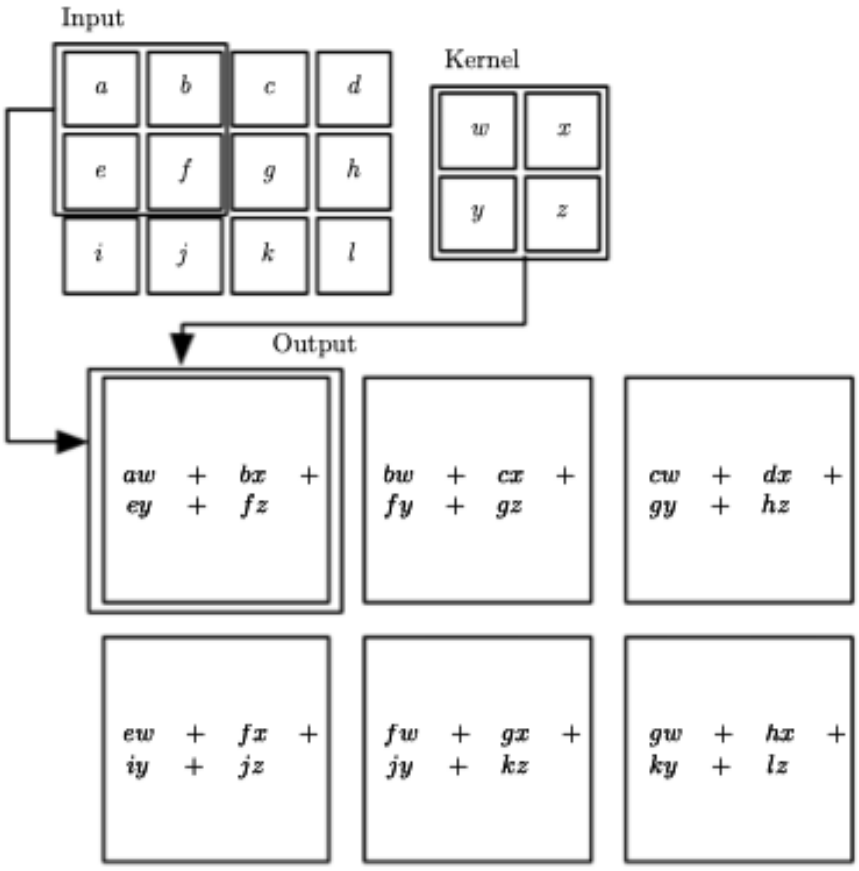


Figure 2.2: Convolution operations in a CNN as shown in [3]. Here, the single-channel (d=1) input image is of dimension 3×4 and filter is 2×2 with stride of 1. Each letter represents a pixel value in both these entities. When the filter is slid over the image, dot products are computed correspondingly (6 times in this case) which form the output.

Rectified Linear Unit (ReLU)

The ReLU function for an input x is defined as below

$$\text{ReLU}(x) = \max(0, x)$$

LeakyReLU is an extension of ReLU where there is a small positive output value even when the input is negative.

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Sigmoid function

It is generally used during binary classification or when we find it essential to convert the output to the range (0,1).

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Softmax function

It is an extension of the sigmoid function used to obtain outputs in form of probabilities for each class during multi-classification tasks.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \text{ where } i = 1, \ldots, K \text{ are output classes, and } \mathbf{x} = (x_1, \ldots, x_K)$$

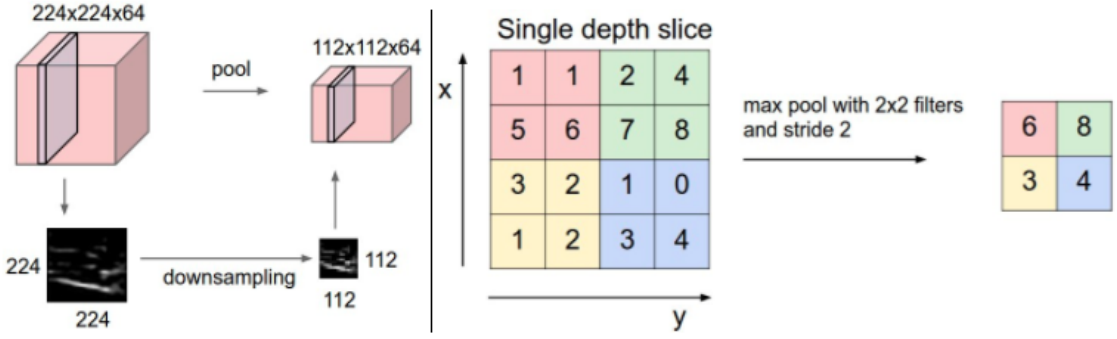


Figure 2.3: The image is taken from [1]. In the left section, pooling operation is done on the input volume of size $224 \times 224 \times 64$ to get an output volume $112 \times 112 \times 64$. Note that the volume is not reduced along the depth dimension (or channels). In the right section, Max-pooling on an activation map is shown. Each output is obtained by taking maximum value across 2×2 windows.

2.1.4 Fully-Connected (FC) Layers

CNNs have Fully-Connected layers towards the end of the model. These layers learn to match the extracted features from convolution layers with their respective correct output. Here, all nodes in this layer are connected to all other nodes in its previous layer. Hence, the computed output of this layer is a matrix multiplication of its nodes and their respective outgoing weights, followed by the addition of a bias term in the end.

2.2 Transfer Learning

The main motive behind transfer learning is to re-use the knowledge gained previously on solving a problem to solve another similar problem. In simple words, we try to transfer the learned ability to recognize patterns within data to perform another task. For example, in a typical classification problem to classify animals present in the image data, the model tries to learn and extract different features of animals to distinguish between them. The learning happens by tuning the model weights which are responsible for feature extraction. The top-most layer then uses these features to classify the image accordingly. But, these extracted features can also be used to solve other similar problems as well, such as the task of recognizing the breed of animal in an image, etc. The features required for solving this task would not be very different from those of animal classification problem. Hence, by trying to re-use those features, we can speed up the process of learning this new task.

Moreover, as the problems involving image data always deals with recognizing simple geometric shapes at its core, the model always learns these shape features using initially present layers. Thus, even if the new task (involving image data) is substantially different from the animal classification problem, the extracted features will still be useful to achieve better and faster results while learning the new task.

2.2.1 Pre-training

We can see pre-training as an extension to transfer learning concept. During transfer learning, the main aim is usually to transfer gained knowledge from a specific task to another similar task. Whereas, pre-training aims to learn all the useful knowledge during the training process so that it can be used for several different tasks [26]. To put it briefly, in transfer learning the model learns features required to solve a specific task and tries to apply this knowledge on any other similar task, while pre-training tries to capture all generic features while solving a (generic) task so that it can also be used for any other tasks in future. Generally, convolution layers are considered to be feature extractors while FC-layers understand the pattern to map the obtained features to its output. Hence, when using a pre-trained network, we usually obtain a feature vector directly from flattened convolution layer nodes and discard the top FC-layers. Currently, there are many pre-trained networks available that are trained on a very large dataset called ImageNet [27]. We will be using two such popular networks during our experiments namely *MobileNetV2* [28] and *VGG16* [29]. In our study, we will be comparing which one of them gives better performance when integrated with `3D_DEN`.

2.3 Literature Review

In this study, we are looking at the problem of open-ended 3D object recognition, which in itself has two sub-problems, namely continual learning and 3D object recognition. Both of these have a deep history of research in machine learning, computer vision, and robotics, resulting in many different approaches. Figure 2.4 displays some of the significant research papers in the form of a Venn diagram (image was taken from [4]). In this section, we review a few of these recent efforts which are most relevant and compare how our work differs, even though it is inspired by them.

2.3.1 Continual Learning (CL) Approaches

Several kinds of approaches have been proposed to the tackle continual learning problem, involving *regularization techniques* for weight changes, *dynamic architectural networks* and *memory replay* [30]. We briefly explore them in the following sub-sections.

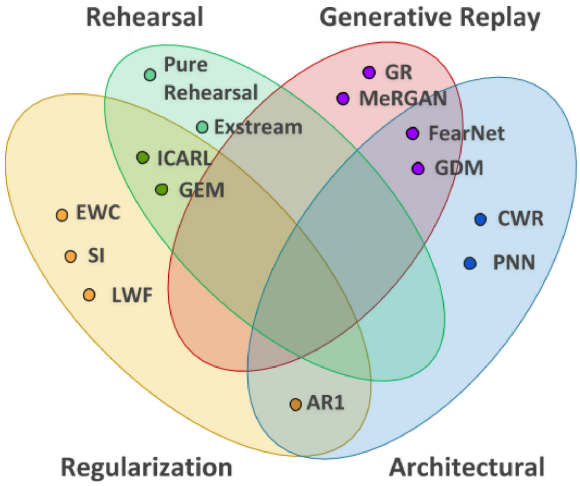


Figure 2.4: This image was taken from the paper [4]. It shows a Venn diagram illustrating recent popular papers on Continual Learning: PNN [5], CWR [6], EWC [7], SI [8], GEM [9], LWF [10], ExStream [11], FearNet [12], ICARL [13], MeRGAN [14], GDM [15], AR1 [16], Pure Rehearsal and GR [17]. Rehearsal and generative replay are subsets of replay strategies [4].

Regularization Techniques

This technique tries to tackle the catastrophic forgetting problem by introducing constraint checks on updation of weight parameters of the model [30]. The use of simple L2-Regularization forbids the model to learn new upcoming tasks efficiently. To overcome this problem, a method called *Elastic Weight Consolidation* (EWC) was proposed in the paper [7] which computes the Fisher information matrix and uses its diagonal to constrain the weight parameters. The paper states that this method identifies which parameters were more important for the previous task and thereby selectively assigns greater constrain on updation of those parameters. This is in contrast to L2-Regularization where we assume

an equal constraint for all the parameters.

Another paper [8], came up with a similar strategy called *Synaptic Intelligence* (SI). In this paper, rather than using diagonals of the Fisher matrix to gain information about variance, the authors tried to find the 'importance factor' of each parameter across every previously learned task. This meant that, while updating the parameters the constraint was not just applied to the change of parameters concerning the immediate previous task. Instead, they took into account the weight changes concerning all previously learned tasks.

X. He et al. [31], tried to use Conceptors [32, 33] to eradicate catastrophic forgetting. Here, the authors treat conceptors as a memory to learn new tasks, where they keep exploiting only the unexplored part of memory to store the incremental weight parameters of new tasks.

Architectural Techniques

This technique tackles catastrophic forgetting by changing the architecture of the model. One such work was the *Progressive Neural Networks* (PNN) [5], where the authors proposed an idea to increase the size of every layer in the model for each new task. Hence, while performing a task, only the existing nodes and newly added nodes that were present while training that task will be responsible to compute the task's output. But, the main drawback was that the model size kept on increasing with new incoming tasks. Also, as all the tasks used the nodes added only till their training stage, only a sub-network from the whole model was used during inference of each task. This leads to significant wastage of storage space [24]. Extending this work, T. Xiao et al. [34] proposed a similar model which not only increases its layer sizes but also branches out towards the upper layers. As described by the authors, the model indeed forms a hierarchical structure to accommodate new categories [34].

Referring to these stated approaches, a new efficient model was recently proposed in the paper [24]. The authors called it 'Dynamically Expandable Networks', which combines both regularization and dynamic architectural methods. This paper has proved that by using the three different training methods, better results can be achieved not only concerning accuracy but also in terms of computational complexity; *Selective Re-training*: here, only a sub-network from the whole model is selected and trained on new upcoming tasks; *Dynamic Network Expansion*: the model is expanded dynamically when the loss obtained in the previous method exceeds a specific threshold. Unused newly added nodes are removed after the training process; *Network Splitting*: if the value of an existing node undergoes a drastic change during training, it is duplicated and this new one is used in the further training process. The authors of this paper have also made sure of efficient memory usage without compromising the overall accuracy. This model was tested on several datasets, including 2D image datasets with smaller dimensions like MNIST [21] and its variations.

One major disadvantage of using all the above-mentioned approaches is that they all assume each task to be a multi-class classification problem. Due to this, during inference, we have to provide a task ID to select the sub-network from the whole model which may then be used to provide output (Section 2.4 discusses this in detail). Not all continual learning problems will have this scenario, such as our problem statement, where each new task is a new upcoming category to be learned. However, these papers form an excellent basis for our work.

Memory Replay Techniques

Memory replay tries to tackle catastrophic forgetting by re-using the old samples belonging to the previously learned tasks. It can further be divided into *rehearsal* and *generative replay* [4]. According to the paper [4], during rehearsal the samples are selected by careful inspection as each set of samples for the previous tasks should thoroughly represent their tasks. However, they can also be chosen randomly. Typically, these samples along with the data of the current task form the coreset [4]. The authors of the paper [9] proposed a methodology called 'Gradient Episodic Memory' (GEM), which makes use of this coreset not only for training but also to apply a constrain on gradient-update while learning new tasks. To be specific, they try to reduce the angle between the loss gradient vector obtained from samples of previously learned task and the current gradient update [9].

The difference in the generative replay is that we make use of generative models known as GANs [35] to create new samples similar to the original samples of previous tasks. This indeed saves a lot of memory by avoiding the need for previous tasks' data, but adds a small computational cost to train generative models. This cost is mainly due to the usage of two models namely, generator and discriminator [35]. As described in [4], the generator has its weights frozen but is used to generate replay samples depicting past experiences. Whereas, discriminator tries to learn from samples of the current task as well as generated replay. When a task is over, the generator model is trained to generate the current task samples, initializing it for the next task.

Further, the generative replay can be subdivided into 'Marginal Replay' and 'Conditional Replay' [4, 36]. According to [4], marginal replays make use of conventional GANs, while conditional replays use GANs which accept specific inputs. This means that these models can generate specific output samples as required based on the given input condition. Hence, they can be used to generate samples of a specific previously learned task, if required.

2.3.2 Object Recognition Approaches

There have been a few interesting approaches specific to tackle the problem of open-ended object recognition. One such method is the instance-based approach proposed in the paper [37]. This demands less training time and works well when we have fewer samples in training data. According to the algorithm discussed in this paper, the model initially con-

siders each new object as a new class and later clusters them together rather than assigning labels to each sample. Even though the authors have proposed this model for unsupervised continual learning in real-world scenarios, it can also be extended for supervised learning tasks with some modifications.

Another similar instance-based learning model called OrthographicNet was proposed by my supervisor Dr.Hamidreza Kasaei in the paper [22]. Here, the model learns interactively, based on three functions: Teach, Ask, and Correct. During training, the model tries to generate a global feature vector for every object category, which is scale and rotation invariant. During inference, the extracted features are compared to these global features to recognize the correct category of the input object.

As described by my supervisor in OrthographicNet [22], initially, three orthographic projection views of an object are obtained. These views are decided based on the principle component axes obtained from the eigenvalue decomposition of the object's point-cloud. Each view of an object is given as input to a separate pre-trained CNN, which was trained on ImageNet [27]. This extracts the required features of each view of the object. Then, these features are merged by element-wise max-pooling to form a single representative feature vector for the given object. During training, when an object from a new category arrives, a global representation for that category is created and initialized. This global representation for the category is updated whenever more objects from this category arrive. In other words, we can say that each category has its feature representation (in lesser dimension) built up using the instances of that category. Finally, during inference, the feature representation of the input object is compared with the global representations of all categories using a similarity distance measure and is assigned the closest category. This model achieves state-of-the-art accuracy in open-ended 3D object recognition scenarios when tested on *Princeton ModelNet40* and *Washington RGB-D Object* datasets [38, 39]. But the main drawback of such an instance-based learning model is that it may only be able to learn a limited number of categories. Another concern may be a decrease in performance when training on similar looking categories. This may indeed be a major concern when the number of known categories becomes very high. Moreover, it also occupies memory to store global representations of all categories along with their object views. This problem is mainly due to the usage of a fixed architectural model with a constant number of neurons for feature extraction and representation. Nevertheless, this work serves as one of the major parent papers that inspired our work.

Another important paper [40] discussed a new model named *PointNet* to recognize 3D objects by taking the input of objects in the form of point-cloud. Although this model cannot deal with open-ended learning scenarios, it serves as a base architecture for many other research papers on open-ended learning. This paper describes three key modules: the max-pooling layer also called symmetric function, a local and global information combination structure, and two joint alignment networks [40]. The Symmetry function makes the model translation invariant and also binds all the information from different points together. The joint alignment network uses an affine transformation matrix on the input

before training, so that the model is capable of generalizing certain geometric transformations while classifying. The local features are aggregated using max-pooling to obtain global features. Finally, a non-linear function transforms it to get the corresponding probabilistic output.

2.4 Comparing 3D_DEN with other approaches

Here, we try to elaborately explain how our work differs from a few of the above-mentioned papers [5, 24] which might seem similar.

The first and most important difference is the type of tasks learned by the model. As stated earlier, our work tries to address the continual learning problem where training a new task means to learn and classify a new category without forgetting already known categories (learned in previous tasks). Whereas the above papers assume every new task to be a multi-class classification that is independent of each other. This means the sequence of tasks to be learned by the model discussed in these papers can be, for example: 'animal classification', 'flower classification' and so on. Also, all these classification tasks will assume to have a fixed number of output categories, making the size of the output layer of the model fixed (for example: 10 types of animals, 10 types of flowers, etc.). Their proposed training methods involve training only a subset of nodes from the whole model to learn a task (discussed in the previous section). They also assign a task ID to the nodes that are currently being trained for each task. This is done to keep track of the set of nodes corresponding to each (independent) task and are responsible for computing its (task's) output. Therefore, during inference, we have to provide a task ID to inform the model to carry out the particular task we want. Then, only those corresponding nodes having this task ID will be selected from the whole model, and form a sub-network to be used for inference. Thus, task ID acts as a timestamp on nodes, that allows users to convey to the model which learned (independent) task needs to be performed.

However, in our work, each task means learning a new object category and to distinguish it from the rest known ones. This makes each task to be dependent or mutually connected to already known tasks. Precisely, we try to tackle the continual learning problem where learning each new task means to incrementally build a single classification model to accommodate more and more new output categories. Thus, in contrast to the fixed-sized output layer used in the above-explained papers, learning a new task in our case, adds a new output node to the model.

Secondly, during *Dynamic Expansion*, the approach discussed in [24] re-trains existing (or old) parameters along with newly added nodes. Whereas, we only re-train the new nodes to prevent catastrophic forgetting. Hence, we do not need a separate method *Network Splitting* discussed in [24] to avoid drastic change in old parameter values.

Thirdly, in our work, we are using a hybrid approach by combining dynamic architecture technique and memory replay (pure rehearsal). Besides, we employ a deep transfer learning approach by using a pre-trained network in our architecture for feature extraction instead of training deep convolution layers from scratch.

Chapter 3

Methodology

Open-ended learning concerns all three issues namely time, computation, and space complexity. Besides, dynamic architectures which we will be using, are known to have very high time/computational cost. However, as our problem at hand is specific to domestic robots, we can assume that the space complexity (storage requirement for data samples) to be less significant. Our main focus is to reduce the computational cost of our dynamic architectural model for faster real-time responses, with little or no compromise in overall accuracy. In other words, we aim to make a model that requires lesser computations during training but gives real-time responses that have acceptable accuracy.

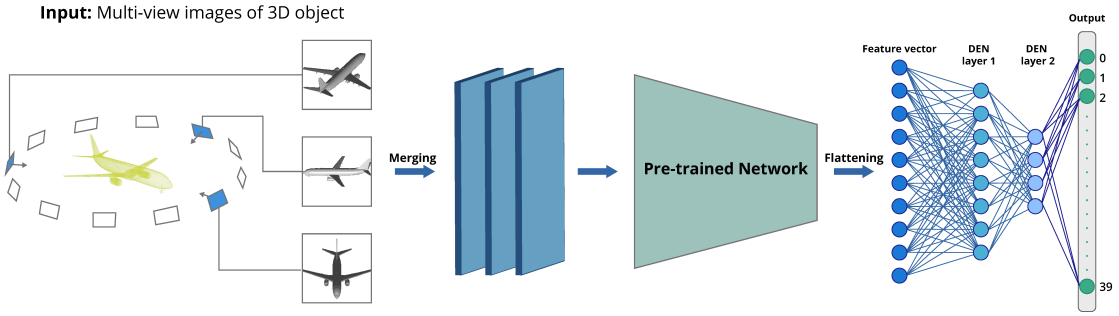


Figure 3.1: This is the proposed architecture of our `3D_DEN` model. Initially, three representative views are chosen from a set of multi-view images for a given 3D object. Then, each of them is converted to a single channel (grey-scale) image and later merged to form a 3-channel image. Now, this image is fed to a pre-trained network, and the extracted features are flattened. Finally, we attach two DEN layers to the model which give the output.

Similar to OrthographicNet, `3D_DEN` takes three different angular view images of each 3D object instance during training. But, rather than having different CNNs for every view image, these view images are converted to grey-scale single-channel images and then combined to form a 3-channel image which becomes our input. This idea of combining views was suggested by my supervisor, Dr.Hamidreza Kasaei. Thus, our overall model design is a modified version of OrthographicNet's architecture, that consist of a single pre-trained

network (with their weights frozen and top-most layer removed) attached to two consecutive trainable DEN layers (this part was proposed by me) which are Fully-Connected (FC) as shown in Figure 3.1.

We believe that the usage of a pre-trained network and having only one CNN will minimize the computational overhead to a great extent. Whereas, a better overall accuracy will be maintained by DEN layers as they will learn new distinguishing features of upcoming categories during training. Throughout the training procedure, weights connecting these layers are kept sparse using L1-Regularization. However, to achieve an absolute sparseness, we also perform a search at the end of each training process, to find weights having very low absolute value and zero them out.

We introduce modern supervised training methods for training these layers, few of them similar to the ones discussed in the paper [24] with modifications as explained in following subsections. Also, all mathematical equations we formulate are inspired (or modified) from the said paper. The learning procedure consists of a series of tasks. The initial task(t=1) is a binary classification of two categories trained according to below Equation 3.1:

$$\underset{W^{t=1}}{\text{minimize}}\, \mathcal{L}\left(W^{t=1}; \mathcal{D}_{t=1}\right) + \mu \sum_{l=1}^{L} \left\|W_l^{t=1}\right\|_1 \tag{3.1}$$

where $\mathcal{L}$ is task-specific loss function, $1 \leq l \leq L$ represents the l_{th} layer in the model. W_l^t represent the weight matrix of the layer l and μ is the parameter for L1-Regularization. Dataset $\mathcal{D}$ contains image views of T number of object categories, where the set of instances for each task (or category) t is represented by $\mathcal{D}_t$. From second task ($t > 1$) onwards, each upcoming task represents the addition of a new unknown category to be learned without forgetting already known ones.

3.1 Data Sampling (Pure Rehearsal)

Ideally, due to the problem of catastrophic forgetting, gradient-based models have to be re-trained again from scratch when newer data becomes available. In our proposed model we try to solve this problem by using the simplest method which is sampled rehearsal of older data along with new data during training each task. This procedure is carried out by sampling the old data for the same number of instances that the new data has. We keep equal sampling probability for all the categories in old data and also maintain a minimum threshold of ρ, on the number of instances to be drawn from an old category during sampling. Algorithm 1 describes our sampling process. The final data for training task t is represented by $\mathcal{D}_t^{'}$. Although we reduced the training data by using sampling, we still have to store the whole dataset during our open-ended learning process. An ideal solution for generic open-ended learning should eradicate this storage problem. We leave this problem for future work.

Algorithm 1: Data Sampling

Input: Dataset $\mathcal{D} = (\mathcal{D}_1, \ldots, \mathcal{D}_T)$, task t (> 1)
Output: $\mathcal{D}_t^{'}$
$s = len(\mathcal{D}_t)/(t-1)$
if $s < \rho$ **then**
  $s = \rho$
end
for $i = 1, \ldots, t-1$ **do**
  Sample s elements from $\mathcal{D}_i$ and add to $\mathcal{D}_t^{'}$
end
Add $\mathcal{D}_t$ to $\mathcal{D}_t^{'}$

3.2 Selective Retraining

When a new task ($t > 1$) arrives, firstly, a new output node o_t (for new category) is added and the weights between top-most hidden layer to the output layer (all output nodes) are retrained using the Equation 3.2. Then, the nodes in this hidden layer that have non-zero weights to the new output node are selected and added to a list S. Further, we perform a layer-wise search among the rest of the sparse DEN-layers (FC) to select all the corresponding nodes which are connected to the previously selected nodes and add it to S. These selected nodes are the ones affected by the new task. Hence, we retrain only them separately as a sub-network using the Equation 3.3. Please note that we also make sure there is no drastic change in their weights by imposing a regularization constraint in the loss function. Figure 3.2 and Algorithm 2 illustrate the working of this method.

$$\underset{W_{L,t}}{\text{minimize}}\, \mathcal{L}\left(W_{L,t}^{t}; W_{1:L-1}^{t-1}, \mathcal{D}_t^{'}\right) + \mu \left\|W_{L,t}^{t}\right\|_1 \tag{3.2}$$

$$\begin{aligned}\underset{W_S}{\text{minimize}}\, \mathcal{L}\left(W_S^t; W_S^{t-1}, \mathcal{D}_t^{'}\right) + \mu \left\|W_S^t\right\|_1 \\ + \lambda \left\|W_S^t - W_S^{t-1}\right\|_1\end{aligned} \tag{3.3}$$

where $W_{L,t}^{t}$ represents the weight tensor of nodes at output layer (L) which is being trained for task t; $W_{1:L-1}^{t-1}$ represents weight tensor of nodes at top-most hidden layer ($L-1$) with values obtained after task $t-1$. W_S^t represents weight tensor of nodes present in the list S, which is being trained for task t; likewise, W_S^{t-1} represents weight tensor of nodes present in the list S after they were trained for task $t-1$. μ is the coefficient of L1-Regularization, and λ is the coefficient of regularization that governs drastic weight change.

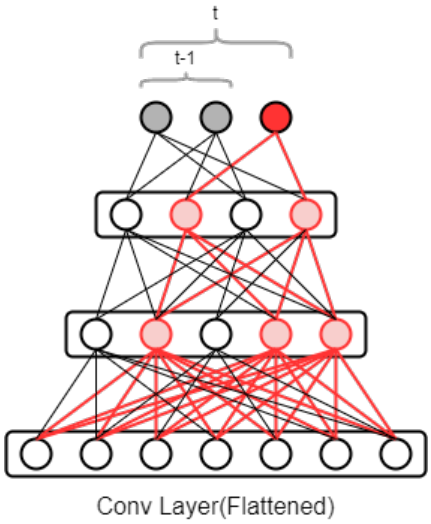


Figure 3.2: Selective Retraining: First, the weights between top-most hidden layer and output layer are trained. In next step, we select nodes in the top-most hidden layer that have non-zero values to the new output node. Then, we perform a layer-wise search to identify rest of the nodes in sparse bottom layers, that have non-zero connections to these selected nodes. Selected sub-network is represented by red color.

Algorithm 2: Selective Retraining (modified version from [24])

Input: $\mathcal{D}_t^{'}$, $\mathbf{W}^{t-1}$
Output: $\mathbf{W}^t$
Initialize $l \leftarrow L-1, S=\{o_t\}$
Solve Eq. 3.2 to obtain $\mathbf{W}_{L,t}^t$
Add neuron i to S if the weight between i and o_t in $\mathbf{W}_{L,t}^t$ is not zero
for $l = L-1, \ldots, 1$ **do**
  Add neuron i to S if there exists some neuron $j \in S$ such that $\mathbf{W}_{l,ij}^{t-1} \neq 0$
end
Solve Eq. 3.3 to obtain $\mathbf{W}_S^t$

3.3 Dynamic Expansion

When the test accuracy A_t (achieved on unseen test samples) falls below a specific threshold (set as τ=0.85) during selective retraining, then new nodes are added to the model to improve performance. Initially, a constant number of new nodes (say k) are added at each layer to train the model for the new object categories. The new nodes have input connections from all nodes from previous layer but their outputs are passed only to new nodes of the next layer. This allows the new nodes to capture new features of the new task category by exploiting information from old nodes without modifying them and thereby preserving the acquired knowledge about older tasks [5]. We only train these new nodes keeping the

rest of the network non-trainable, using the below Equation 3.4:

$$\underset{W_l^{\mathcal{N}}}{\text{minimize}}\, \mathcal{L}\left(W_l^{\mathcal{N}}; W_l^{t-1}, \mathcal{D}_t^{'}\right) + \mu \left\| W_l^{\mathcal{N}} \right\|_1 \tag{3.4}$$

where $W_l^{\mathcal{N}}$ represents the weight tensor of newly added nodes at layer l; W_l^{t-1} represents the weight tensor for nodes at layer l with its values obtained after task $t-1$. Any new node that seems to be useless after training a task is later removed. We assume a node to be useless when the absolute values of all of its output weights are lesser than a specific threshold ε (set empirically). This makes sure that both the training procedure as well as the size of the model's architecture are optimized with regards to computation cost, at the end of the learning process. Figure 3.3 and Algorithm 3 illustrate the working of this method.

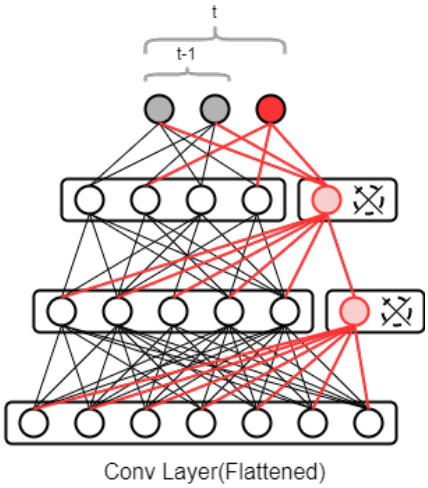


Figure 3.3: Dynamic Expansion: New nodes are added to DEN layers and trained, while rest of the nodes are kept constant. In the end, new nodes that were found useless are removed. Here, newly added nodes are represented by red color.

Algorithm 3: Dynamic Expansion (modified version from [24])

Input: $\mathcal{D}_t^{'}$, ε
Output: $\mathbf{W}^t$
Add k neurons $h^{\mathcal{N}}$ in all layers
Solve for Eq. 3.4 for all layers
for $l = L-1, \ldots, 1$ **do**
  Remove useless neurons in $h_l^{\mathcal{N}}$ whose output weights are less than ε
end

3.4 Dataset and Pre-processing

We used the *Princeton ModelNet40* dataset [38] for performing all our experiments. It contains 12,311 CAD models from 40 different object categories, which were divided into 9,843 training samples and 2,468 testing samples. Image view format of this dataset was obtained from the paper [41]. Here, each object sample is represented by a set of 12 image views that are obtained by setting the camera around the object at different angles on the horizontal plane of the object (angular difference of 30° from each other w.r.t object). Each view was given a unique ID (value between 1-12) to represent its respective camera angle.

By manually inspecting a few such samples, we realized which set of camera positions will be sufficient enough to describe the shape of the object. We observed that the image views having camera positions which were approximately 120° (a circle that lies on a plane forms 360°) apart from each other, best described the shape of the objects. Thereby, we identified and selected the three best representatives (using their unique ID) out of 12 views for every object, which had the above stated angular difference. Also, note that each view image was converted to a single channel (grey-scale) with dimensions 128×128. The starting step illustrated in Figure 3.1 might give a better picture of how we obtain the best representative views. This is how we ultimately formed our final dataset (includes both train and test data). The above pre-processing step was performed to reduce the computational cost of the training process.

Chapter 4

Experiment and Results

Three types of experiments were carried out to evaluate our proposed model. First, we present a systematic open-ended evaluation of the proposed `3D_DEN` approach in the context of the object recognition task. Second, we perform an offline evaluation using a similar architecture as `3D_DEN`, but with a fixed size of FC-layers. Lastly, we also performed a real-robot demonstration in the context of `serve_a_drink` scenario to show the strength of the proposed approach concerning real-time performance. In the following subsection, for each type of experiment, we first describe the experimental setup and then discuss the obtained results.

4.1 Open-Ended Training Evaluation

In this round of experiments, we evaluated three different pre-trained networks to find out the best architecture for our `3D_DEN` model in terms of accuracy and computational cost. Two of the networks are popular feature extractors namely VGG16 [29] and MobileNet-v2 [28], both pre-trained on ImageNet [27]. The third network is a custom model built based on the architecture of MobileNet-v2, and trained on *ModelNet10* dataset [38] which later serves as a pre-trained network for this experiment. The intention of considering various pre-trained architectures was to find if there is any notable difference in feature extraction when the input for pre-training is not RGB images but rather three single-channel view images. Table 4.1 gives more insights on these networks' properties. Towards the end, we compare the best `3D_DEN` variant to OrthographicNet, and discuss which approach

Table 4.1: Properties of Feature Extractor Networks used in Experiment

Model	Feature Length	Depth
VGG16	$4 \times 4 \times 512$	23
MobileNet-v2	1280	88
Custom	1280	88

yields better results.

The training of the model is performed in a supervised manner where the input is a 3-channel image obtained by combining three grey-scale views of an object (as described in the previous chapter), while the label of the object is fed as output. The model makes use of the training methods described in the previous Chapter 3 to learn these input representations and match with the outputs. The optimization happens through a gradient-based stochastic optimizer called Adam [42]. We kept the default values for all the hyper-parameters used in this optimization technique.

Adam [42] combines two popular methods, namely AdaGrad [43] and RMSProp [44]. As described in the paper [43], AdaGrad uses different learning rates (η) for different trainable parameters. It assigns higher values of η to parameters which give less frequent features, and lower values of η to parameters which give more frequent features. This is well-suited for training on sparse input (like images) as each parameter can have its own value of η. Whereas, RMSProp tries to dampen the unwanted oscillations during gradient descent [44]. Due to the above advantages, Adam achieves much faster overall convergence than other gradient-based methods, compelling us to use it in our work.

Since the order of introducing tasks may influence the performance, we performed 10 trials for each of the pre-trained networks. In each trial, the model was trained from scratch on the ModelNet40 dataset in an open-ended fashion and overall accuracy was noted down. Every trial consisted of 39 tasks, i.e., one less than the total number categories because the initial task is a binary classification. The order in which the categories appeared for training was random in every trial. Algorithm 4 describes this process in a step-wise manner. After the completion of trials, measurements based on different metrics were computed.

It should be noted that the classical form of evaluation which considers accuracy as the main metric cannot be used for open-ended performance evaluation. This is because, in open-ended evaluation, training happens continually and therefore the final accuracy obtained in the end cannot be the only criterion that defines the efficiency of the training methods. Hence, we consider three main metrics introduced by the papers [45, 22, 46] to compare and discuss the performance of our models. These metrics include *(i)* Global Classification Accuracy (GCA), which computes the average of final accuracies for all our trials; *(ii)* Average Protocol Accuracy (APA), which describes the average accuracy over all tasks and all trials; *(iii)* Average number of Learned Categories (ALC) during each trial. In OrthographicNet, a trial is stopped as soon as the test accuracy falls below a given threshold (set as 66.7%). Therefore, each trial may learn a different number of tasks, and thereby ALC value is computed. But, in our approach we do not have such a stopping criterion, rather we continue to train the model till all categories (tasks) are learned during each trial. Thus, the ALC value for all our models is 40, which intuitively means that each trial involved open-ended training of all (40) categories. This is done to compare the full capability of each model in learning new tasks and decide which performs the best.

This ALC criterion was used in our evaluation only to compare how our 3D_DEN variants perform with respect to OrthographicNet. Additionally, to evaluate the computational cost of training a model, we consider the total number of parameters in the model.

Algorithm 4: Training Procedure (inspired from [24])

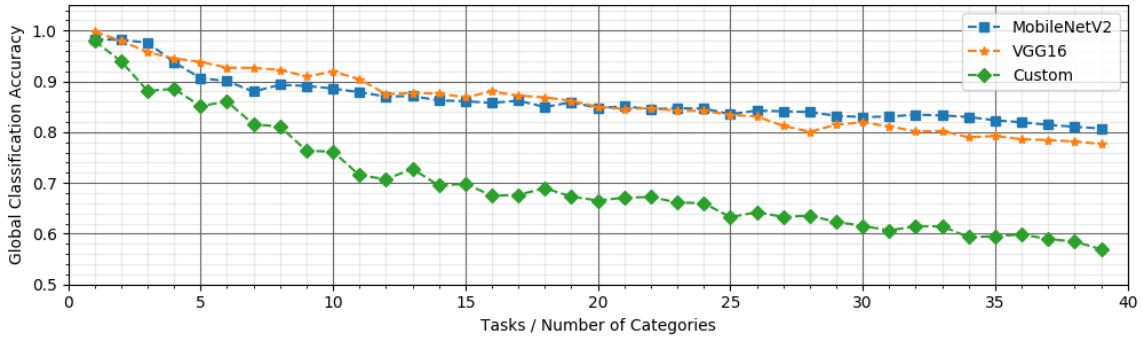
Input: Dataset $\mathcal{D}=(\mathcal{D}_1,\ldots,\mathcal{D}_T)$
Output: $\mathbf{W}^T$
for $t=1,\ldots,T$ **do**
  if $t=1$ **then**
    Train the network weights $\mathbf{W}^1$ using Eq. 3.1
  else
    $\mathcal{D}_{t'} = DataSampling(\mathcal{D},t)$
    $\mathbf{W}^t = SelectiveRetraining(\mathbf{W}^{t-1})$
    if $A_t<\tau$ **then**
      $\mathbf{W}^t = DynamicExpansion(\mathbf{W}^t)$
    end
  end
end

Results and Interpretations : During our extensive set of open-ended training with different settings in the experiment, we compared the performances of our 3D_DEN models with OrthographicNet (current state-of-the-art). Table 4.2 shows the results. We can see that our 3D_DEN variant with Mobilenetv2 as its pre-trained network performed the best in terms of both the accuracies namely GCA and APA, making it the new state-of-the-art model for open-ended evaluation on this dataset. Whereas, OrthographicNet was able to learn only 38 categories after which it had to be stopped as it reached the saturation point. In particular, our model achieved 80.71% as GCA and 84.16% as APA on the test data while using only one-third of the parameters of OrthographicNet. Thus, it not only gives better results but reduced the computational cost to a great extent.

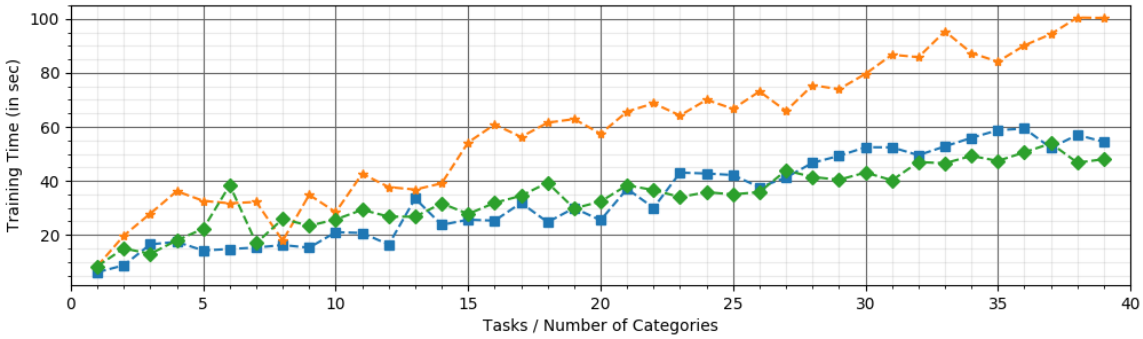
Table 4.2: Result of Open-Ended Evaluation on ModelNet40 Dataset

Model	GCA(%)	APA(%)	ALC	#Parameters
OrthographicNet [22]	66.54	74.70	38	$3 \times 3.53 = 10.59$ M
3D_DEN (ours-*VGG16*)	77.60	83.92	40	138.35 M
3D_DEN (ours-*MobileNet*)	**80.71**	**84.16**	40	**3.53 M**
3D_DEN (ours-*Custom*)	56.93	68.21	40	3.53 M

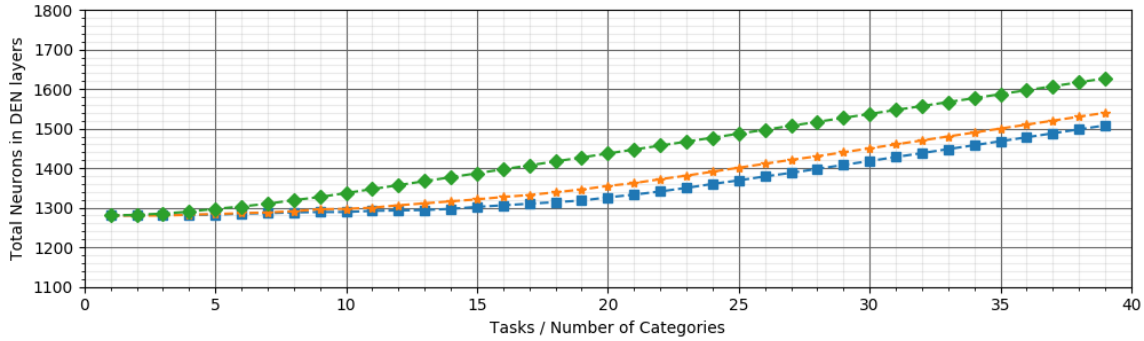
Figure 4.1 shows the timeline graph of GCA (*top*), training time (*middle*) and increase in number of neurons in DEN layers (*bottom*) while learning newer tasks. In the *top* plot we can see that as the newer tasks are being learned, GCA for all the models started to



(a)



(b)



(c)

Figure 4.1: Summary of open-ended evaluation: (*top*) shows the timeline of GCA for all three models; (*middle*) shows training time (approx) needed by our models while learning new tasks; (*bottom*) shows the increment in number of neurons in DEN layers for all models.

decrease gradually. This was an expected phenomenon because when newer knowledge is being gained, the chances of remembering all learned tasks reduces. To put it technically, as the learning progresses, the weight matrices of the model gradually adapt (or overwrite) themselves to learn newer categories thereby forget to recognize features learned from previous categories. Secondly, we do see some overlapping between the curves of `3D_DEN_MobileNet` and `3D_DEN_VGG16` models. Also, by referring to APA values from Table 4.2, both of them performed somewhat similar but differed in terms of GCA values. Whereas, our `3D_DEN_Custom` model performed the poorest of them all.

The *bottom* plot illustrates the training time for each task for all the models. Please note this calculation only considers the training time of DEN layers(i.e. trainable parameters) excluding feature extraction time from pre-trained networks. Moreover, this time calculation also depends upon other background processes running on the system, thus making this plot only an approximation to get an overview of the computational complexity of the models. Having said that, we do observe the training time of `3D_DEN_VGG16` to be much higher than the other two models which can indeed be explained by the size of extracted feature vectors used by the models. The *bottom* plot shows a gradual increase in the size of DEN layers with newer tasks. With precise examination, we also observe that `3D_DEN_Custom` starts to add newer neurons in the DEN layer much earlier than other models, which in-turn depicts that it was unable to learn new tasks by training with existing neurons. To put it in technical terms, existing neurons proved insufficient to gain newer insights from the extracted features while learning almost every new task. Hence, this model had to use the second approach that is Dynamic Expansion at a very early stage.

Another major observation here is the slope of the *top* plot, which looks a bit steep towards the start but later this steepness decreases with upcoming tasks. With further inspection, we can see that each model has a threshold value on the x-axis after which the steepness decreases more significantly. To interpret the reason behind this phenomenon, lets consider `3D_DEN_VGG16` and `3D_DEN_MobileNet` for which this threshold is around *task-15* in x-axis. By referring to *bottom* plot which describes the increment of neurons during training, *task-15* seems to be the point after which both these models start to add neurons to its DEN layers. This indeed conveys that our second approach which is Dynamic Expansion, yields better results than selective retraining. Even though both seem to suppress catastrophic forgetting, the latter seems to do the job more efficiently.

4.2 Offline Evaluation using Grid Search

Through this evaluation procedure, we try to find an approximate number of neurons in FC-layers and the appropriate optimizer required for a model that can achieve the best accuracy during offline training on our dataset. Our main intention was to compare these results with our best performing `3D_DEN` variant (`3D_DEN_MobileNet`) which was trained in an open-ended fashion. Precisely, we will observe how the DEN layers of `3D_DEN_MobileNet` differ from the optimal size of FC-layers in the model obtained here and also seek to find if there are any differences in optimizer used.

In this round of experiments, the model training is governed by the Grid Search algorithm. Initially, the parameter range for FC-layer size and set of optimizers to be used (refer Table 4.3) are fed to grid-search. Then, we perform a series of training, each time using the whole dataset and using a new parametric configuration (obtained from grid-search) for the model. In simple terms, each training procedure involves building a model according to a configuration obtained from the grid-search algorithm and training it from

scratch on the whole dataset in an offline fashion (on all 40 categories at once). The idea is to inspect which model configuration performs the best as each training procedure has a different combination of FC-layer sizes and optimizer. Table 4.3 shows the best configuration results obtained, along with the input parameters of grid-search. The figure 4.2 shows all these configurations with their respective accuracies obtained. The top three configurations based on accuracy are shown in different colors (green, red, and purple) while the rest are represented in light-grey. The green-colored connection shows the best configuration.

Table 4.3: Grid Search Parameters for Offline Evaluation

Grid Parameters	Range	Best Parameters	Best Accuracy(%)
FC-layer:1	{512,1024,2048}	2048	90.72
FC-layer:2	{128,256,512}	512	
Optimizer	SGD, Adam	SGD	

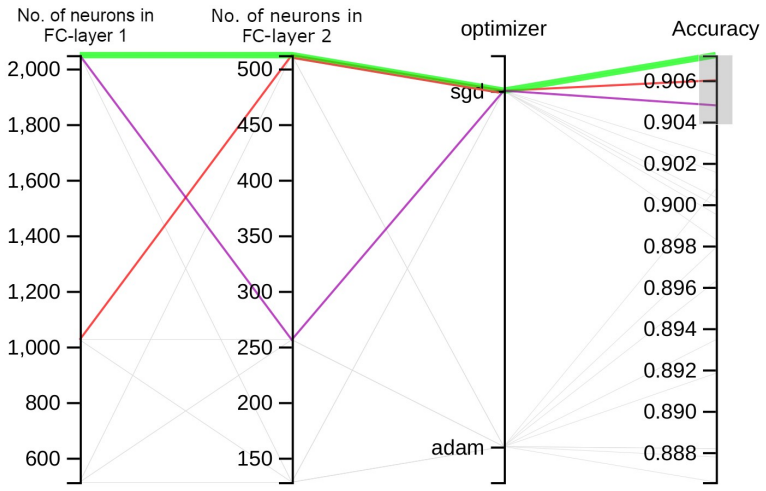


Figure 4.2: Results of offline evaluation using grid search is shown above. The best parametric configuration is represented by green-colored connection. Whereas, the red and purple-colored connections show the parametric configurations which achieved second best and third best results, respectively.

Results and Interpretations: Table 4.3 and Figure 4.2 shows the results of the offline evaluation, where we can see that the best size of FC-layers were 2048 and 512 (so a total of 2560), yielding the highest accuracy of 90.72%. Now, by comparing these results to `3D_DEN_MobileNet` (referring to Figure 4.1-*bottom*), we observe that our model used only around 1500 neurons in total for both DEN layers which indeed seems reasonable when we correlate them with the respective accuracies they achieved. Also, to our surprise, we observed that offline evaluation gave the best accuracy using SGD optimizer, whereas open-ended evaluation models always gave the best results using Adam optimizer.

4.3 Evaluation on Real-Time Robot

A real-robot experiment was carried out to check the performance of the proposed model in real-time. Precisely, this experiment aimed to see if the robot can recognize a set of table-top objects using our trained `3D_DEN` model to accomplish a given task. In particular, the task here was to pour a drink from a bottle into cups present on the table. The experimental setup is depicted in Figure 4.3. It mainly consists of a Kinect sensor and UR5e robotic-arm that perceives the environment and acts accordingly. There are four instances of three object categories on the table: two cups, a bottle, and a vase object with flowers. This is a suitable set of objects for this test since similar instances to these selected objects exist in the ModelNet40 dataset.

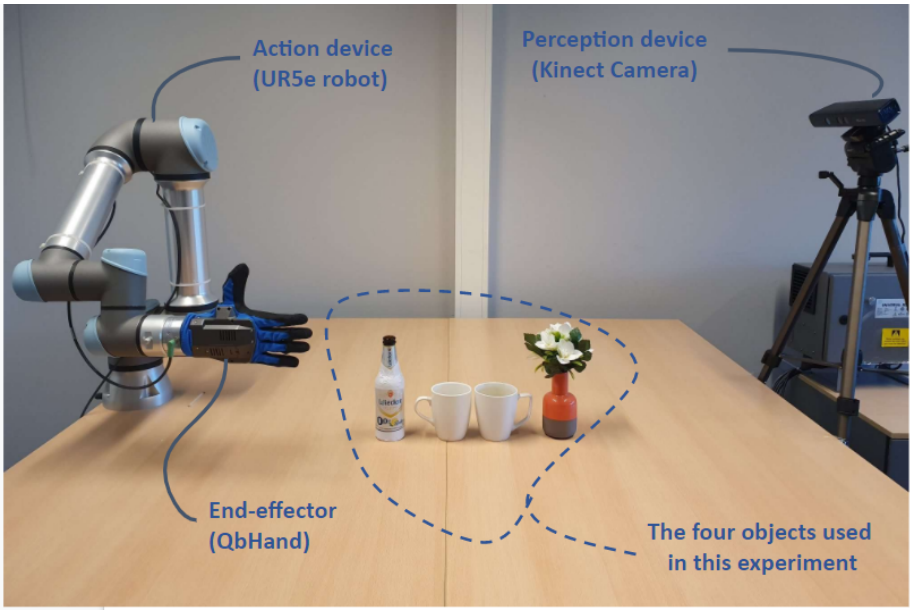


Figure 4.3: Our experimental setup for real-robot experiment consists of a Kinect sensor to perceive the environment, and a UR5e robot to act upon the environment. Throughout the `serve_a_drink` experiment, we use four instances of three object categories including *bottle*, *cup*, and *plant*. It should be noted that similar instances to the selected objects exist in the ModelNet40 dataset.

The cognitive robotic system used in this experiment was built based on the paper [18], written by my supervisor Dr.Hamidreza Kasaei (as the first author). The architecture of this whole system (taken from [18]) can be seen in figure 4.4. As described in this paper, the system pipeline consists of various modules namely, Object Detection, Object Recognition, Grasp Affordance Recognition (based on pose), and Grasping. In this experiment, we used our final trained `3D_DEN_MobileNet` model obtained from open-ended evaluation to replace the existing Object Recognition module (highlighted as a red box in the figure 4.4). The rest of the modules were kept unchanged.

We now explain the working of each of these modules as described in [18]. The Object Detection (and Tracking) module described in [18] combines the methods proposed in the

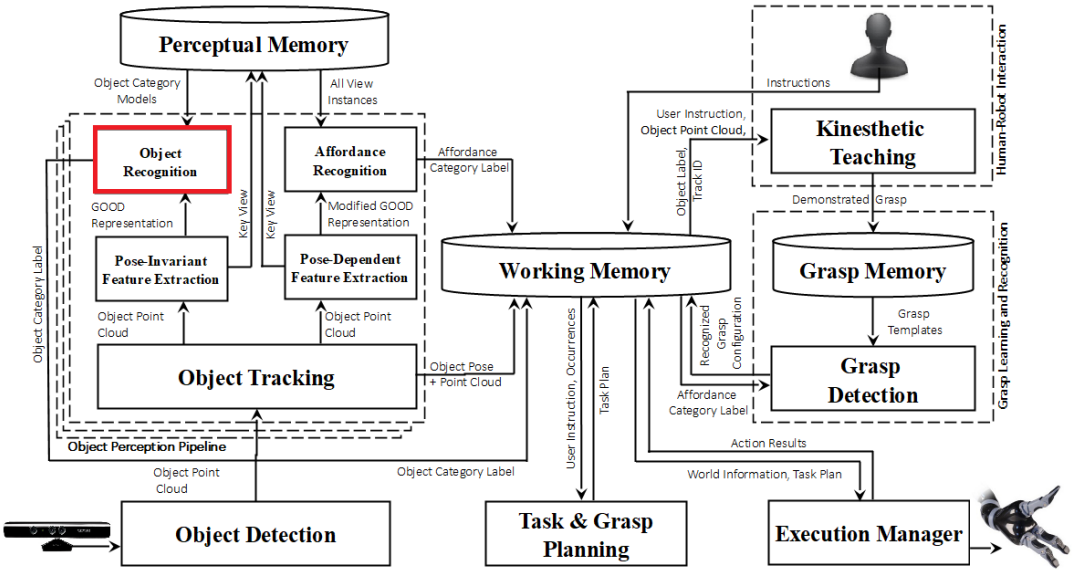


Figure 4.4: Image taken from [18]. Each depicted module in the diagram is run as a ROS [19] service and the connections show the flow of information between them [18]. The highlighted red module is replaced by our `3D_DEN`.

papers [47, 48]. Here, initially, the point cloud data describing the scene is collected from the 3D-sensor camera (fixed at a location). If a region in this point cloud has points that are located closely in continual fashion (on all the 3 dimensions), then they are considered as an object. The algorithm proposed in [48] is applied to grow this region by merging more points that closely resemble the existing points in the region. Each resultant region (or cluster) formed in this way represents an object and is given a tracking ID [18]. Each cluster is passed to a sub-module named 'Pose-invariant Feature Extraction' module shown in figure 4.4. This module contains the entity called 'Global Orthographic Object Descriptor' (GOOD) [49] which computes three orthographic projection views of the object cluster using its assigned reference frame. These views are converted to grey-scale and stacked together to form 3-channel image input for our model. The output of our model is used to label the input object. These label information for every detected object is stored in the working memory.

As described in [18], to execute grasping, the point cloud of each object is first passed to 'Pose Dependent Feature Extraction', which is a modified version of GOOD descriptor [49]. This sub-module computes a pose-specific feature vector based on the pose of the object using orthogonal object views and sends it to the Affordance Recognition module. Affordance categories can be defined as the set of possible actions using which an object can be grasped given its pose. It should be noted that the object categories do not relate directly to affordance categories. Indeed objects from the same category can have different affordances for grasps and vice-versa. While teaching affordances, all those objects having similar pose or pose-specific features that match to the same affordance category, are merged to form a representative feature vector of that affordance category, which is

then stored in Perceptual Memory. Therefore, while inferring from the Affordance Recognition module, the pose-specific feature vector for the object is computed and compared with every representative feature vector of grasp affordances using a similarity measure. The closest category is taken as the object's grasp affordance. This information is stored in grasp memory.

The Grasping module computes a grasp point based on the affordance category of the detected object [18]. A comprehensive explanation of this can be found in these papers [18, 50, 48]. Based on this grasp point, the end-effector arm tries to grasp a specific object as instructed in the task.

(a) (b) (c)

Figure 4.5: Comparison between a real-world object with the instances used for training: (a) shows the partial point cloud of the vase object captured by the Kinect sensor in our experiment; (b) and (c) show a vase and plant instance from the ModelNet40 dataset, on which our model was trained.

Results and Interpretations: To accomplish the `serve_a_drink` task, the robot should first detect all the table-top objects, recognize their respective labels, and then decide which affordance action should be performed on objects based on their pose. Afterward, it has to grasp the bottle object and transport it on top of each active cup and serve the drink. The robot should finally return to the initial pose. Towards this goal, the robot first segments the point cloud of each object from the scene and then computes three single-channel orthographic views for each of the objects. Then, the obtained orthographic views of each object are sent to the `3D_DEN_MobileNet` as a service request, and the recognition results are received as the service response (output) in 30Hz. Figure 4.6 shows a sequence of snapshots of the outputs of object recognition and manipulation while the robot performing serve_a_drink task[1]. While experimenting with the robot, we observed that the robot was able to precisely detect the pose and recognize the label of all objects most of

[1]A video demonstration is available at https://youtu.be/tf4trRMyQ0Y

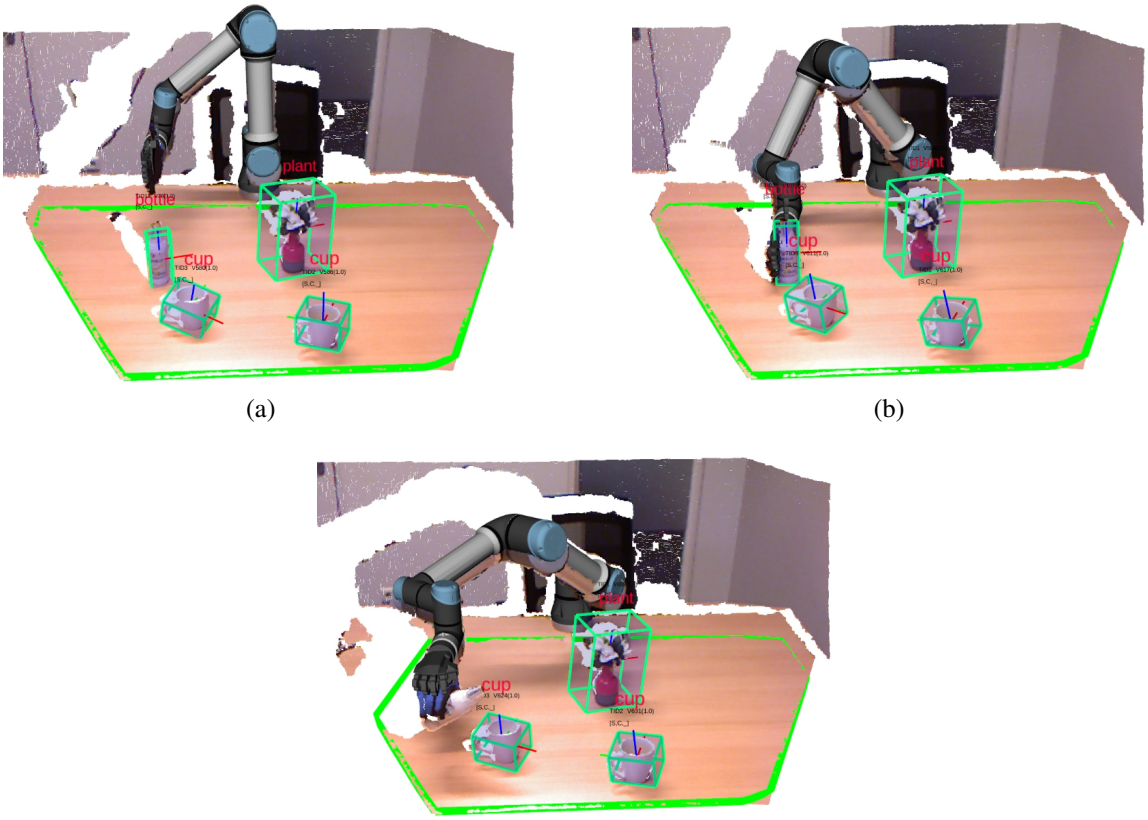


Figure 4.6: A sequence of snapshots from the real-robot evaluation: (a) This snapshot shows our model `3D_DEN` being used to recognize the objects present on the table; (b) The robot grasps the bottle after recognizing it; (c) The robot manipulates the bottle on top of the cup and tries to pour a drink from the bottle into the cup. In these images, the pose of objects is shown by green 3D bounding boxes, and the recognition results are displayed on top of the objects in red.

the time. We also observed that the inferred label for the vase object kept changing between '*vase*' and '*plant*'. This classification error indeed makes sense, because both vase and plant categories from our dataset (on which texttt3D_DEN_MobileNet was trained) look very similar to the vase real-world object. Figure 4.5 shows the comparison between the vase object used during our experiment with vase and plant instances from our dataset. However, given the good performance of our object recognition model, the robot was able to recognize and grasp the bottle and pour a drink from it into the cups present on the table.

Chapter 5

Discussion and Conclusion

5.1 Discussion

Even though our model performed well in the experiments, it also gave insights on some shortcomings guiding us to work on them in the future. We will now explore each of them in detail.

5.1.1 Constant increase in DEN size

The most important one is regarding the dynamic increase in DEN layers. From the *bottom* plot, we can see that after a particular task number, the number of neurons kept increasing at a constant rate. This shows that after adding new neurons, useless ones are not being removed at the end of every task. This heavily contradicts our expectation that our approach will remove all of the unwanted newly added neurons. Nevertheless, when we closely inspected the increment of DEN layer sizes in each trial of the experiment, we did observe some rare occasions in which our mechanism eliminated the useless neurons. This drawback also conveys that in the Dynamic Expansion approach, there was not much sparsity achieved in the weight matrices of newly added neurons. This indeed leads to the conclusion that our usage of L1-Regularization alone is not sufficient enough to achieve sparsity while learning continually. Sparsity among new neurons is essential because, otherwise, every neuron will be trained without any restriction to learn all upcoming tasks, thereby becoming an essential part of the model.

5.1.2 Inefficiency of Selective Retraining

The same plot gives us one more major insight into our model. The model is essentially unable to use our Selective Re-training approach after a particular task number, though it is backed up by our second more efficient approach which is Dynamic Expansion. We can see from figure 4.1(a) that the accuracy falls a bit steeper each time while performing selective retraining, and eventually fails after a particular task. This shows that even while retraining only the sub-network of the existing model makes it forget some of the learned

tasks. It should be noted that this result was obtained even after using regularization constraints on the model weights.

5.1.3 Inability to learn color features

Another problem is with the converting of view images to single channels and combining them. Although it reduced the computational cost of our model to a great extent, it did incur a limitation to our approach, which is the inability of our model to learn the color features of 3D object. Moreover, as the model demands input in such a way, there is a small additional computational cost for pre-processing here. Having said that, if this computational cost becomes a lesser concern in certain scenarios due to the advancements in hardware, then the performance of the model can be improved further by using more number of views than just three.

5.1.4 Memory requirements

Lastly, let's come to the storage problem which our model does not address. Even though our model uses *data sampling* to reduce the rehearsal training data, this still means that the dataset has to be stored during the whole learning process. In contrast to our sampling method, the training procedure of OrthographicNet [22] is governed by a teaching protocol named *simulated teacher* which interacts with the learning agent (neural network model). This protocol uses three main functions: `Teach-Ask-Correct`. The main purpose of these functions is to teach, test, and correct the model when it makes mistakes in recognizing a particular object category. Hence, the rehearsal happens here only when required and not on a fixed basis like in our approach. Nevertheless, in both cases, the dataset always has to be stored in memory. We seek to find a solution to this problem in the future.

By considering the above-stated issues and noticing the trend of gradually decreasing the accuracy curve in our open-ended training experiment, we can say that the model has to be eventually stopped from learning after a point to maintain a certain level of accuracy. With these drawbacks in mind, we can say that our model is not capable of learning a very large number of tasks as we wanted it to be. But, these observations will prove to be vital while extending our work in the future.

5.1.5 Answers to research questions

Which `3D_DEN` model performs better?

From the results we got, it can be seen that `3D_DEN_MobileNet` outperforms the rest. It also sets the benchmark as a new state-of-the-art. It has not only achieved better accuracy but also has lesser computational complexity even while using an architectural adaptation technique. However, as stated before, its limitation is that it cannot learn a very large number of tasks. This means it does not have a significantly high learning capacity. Indeed, this drawback is in all our `3D_DEN` models in general.

How does the best `3D_DEN` model compare with Offline-trained model?

The comparison showed that the number of neurons in the DEN layers of `3D_DEN_MobileNet` is relatively lesser than the FC-layers of the offline-trained model. This disparity can be considered as a result of the difference in the accuracy achieved by both the models. This shows that our approach did indeed add only a reasonable number of neurons during training.

Another interesting distinction between the two was the optimizers used. The best optimizer for the offline-trained model was SGD, in contrast with `3D_DEN_MobileNet` which achieved the highest performance using Adam.

How does the best `3D_DEN` model perform in real-world scenarios?

This evaluation was mainly conducted to check whether our `3D_DEN` model adapts efficiently to the real-world tasks. This ensured that even though the model was trained on a different dataset of 3D objects, it was still able to generalize to real-world objects that were similar to the ones in the training data. From the interpretation of the experimental results, we can conclude that our model did generalize well on data directly coming from the real-world. Even though the model kept changing the label of 'vase' object between *vase* and *plant*, this indeed made sense given the disparity arose due to training data images.

5.2 Conclusion

In this paper, we proposed a deep learning based approach named `3D_DEN` that makes use of dynamic architectural design to learn 3D objects in an open-ended fashion. This approach not only achieves better results in terms of accuracy but also proves to be very efficient in terms of computational complexity, which is indeed considered a major concern while using dynamic architectures in general. While this model can be seen as a new state-of-the-art benchmark, it should also be noted that this model paves a new path (by using dynamic architecture) for solving the problem of continual learning using 3-dimensional data in robotics domains. It can indeed also be considered in other domains that use 3D data, but of course, with some improvements/changes to adapt to those domains.

5.2.1 Future Work

Even though we did achieve satisfactory performance using our model, there were few areas which needs improvement. As we discussed in the above section, one major flaw is the control of the dynamic increase in DEN layers. We seek to solve this issue by coming with a better optimization strategy to reduce the usage of neurons. Moreover, as of now, we are adding a constant number of neurons (hardcoded) during Dynamic Expansion. Deciding appropriate values for other hyper-parameters(and certain thresholds) is one more concern. In the future, we will try to find a strategy that will decide these values using one more learning-based approach. Indeed, there exists a paper [51] which proposed a method

using reinforcement learning to predict optimal values for these hyperparameters. This seems to be a good starting point for further improving our work.

As said earlier, storage and rehearsal of data is one other major concern. There have been few notable papers to eliminate the rehearsal strategies by making use of generative approaches [35, 52]. One such latest work that is also similar to our work (to some extent) is the paper [53] which seems to give promising results by combining various recent strategies to mitigate catastrophic forgetting. Another relevant brain-inspired implementation can be seen in paper [54], which neither uses a generative approach nor attention mechanisms to address continual learning problem. In the future, we seek to integrate one or more of these above strategies in our work.

Bibliography

[1] Li Fei-Fei, Krishna Ranjay, and Xu Danfei. Cs231n convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks, 2014. Lecture notes; accessed on 01-November-2020.

[2] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018. Online; accessed on 01-November-2020.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[4] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz Rodríguez. Continual learning for robotics. http://arxiv.org/abs/1907.00182, 2019.

[5] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. http://arxiv.org/abs/1606.04671, 2016.

[6] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78, pages 17–26, 2017.

[7] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[8] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[9] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476, 2017.

[10] Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016.

[11] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776. IEEE, 2019.

[12] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018.

[13] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[14] Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. In *Advances in Neural Information Processing Systems*, pages 5962–5972, 2018.

[15] German I Parisi, Jun Tani, Cornelius Weber, and Stefan Wermter. Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in neurorobotics*, 12:78, 2018.

[16] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116, 04 2019.

[17] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[18] S. H. Kasaei, Nima Shafii, Luís Seabra Lopes, and Ana Maria Tomé. Interactive open-ended object, affordance and grasp learning for robotic manipulation. In *2019 IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, pages 3747–3753. IEEE, 2019.

[19] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. `https://www.ros.org`.

[20] S. Thrun. A lifelong learning perspective for mobile robot control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 1, pages 23–30 vol.1, 1994.

[21] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 2010.

[22] S. H. Kasaei. OrthographicNet: A deep learning approach for 3D object recognition in open-ended domains. *CoRR*, abs/1902.03057, 2019.

[23] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165, 1 1989.

[24] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *CoRR*, abs/1708.01547, 2017.

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[30] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. `http://arxiv.org/abs/1802.07569`, 2018.

[31] Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. In *International Conference on Learning Representations*, 2018.

[32] Herbert Jaeger. Controlling recurrent neural networks by conceptors. `http://arxiv.org/abs/1403.3369`, 2014.

[33] Herbert Jaeger. Using conceptors to manage neural long-term memories for temporal patterns. *The Journal of Machine Learning Research*, 18(1):387–429, 2017.

[34] Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, pages 177–186, 11 2014.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[36] Timothée Lesort, Alexander Gepperth, Andrei Stoian, and David Filliat. Marginal replay vs conditional replay for continual learning. In Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*, pages 466–480, Cham, 2019. Springer International Publishing.

[37] Luca Erculiani, Fausto Giunchiglia, and Andrea Passerini. Continual egocentric object recognition. `https://arxiv.org/abs/1912.05029`, 2019.

[38] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3D shapenets for 2.5D object recognition and next-best-view prediction. `http://arxiv.org/abs/1406.5670`, 2014.

[39] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *2011 IEEE International Conference on Robotics and Automation*, pages 1817–1824, 2011.

[40] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. `http://arxiv.org/abs/1612.00593`, 2016.

[41] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. ICCV*, 2015.

[42] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[43] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[44] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[45] M. Oliveira, L. Seabra Lopes, G. H. Lim, S. H. Kasaei, A. D. Sappa, and A. M. Tomé. Concurrent learning of visual codebooks and object categories in open-ended domains. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2488–2495, 2015.

[46] S. H. Kasaei, Lués Seabra Lopes, and Ana Maria Tomé. Coping with context change in open-ended object recognition without explicit context information. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–7. IEEE, 2018.

[47] S. H. Kasaei, Juil Sock, Luis Seabra Lopes, Ana Maria Tome, and Tae-Kyun Kim. Perceiving, learning, and recognizing 3D objects: An approach to cognitive service robots, 2018.

[48] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch volumes: Segmentation-based consistent mapping with rgb-d cameras. In *2013 International Conference on 3D Vision - 3DV 2013*, pages 398–405, 2013.

[49] S. H. Kasaei, L. Seabra Lopes, A. M. Tomé, and M. Oliveira. An orthographic descriptor for 3D object learning and recognition. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4158–4163, 2016.

[50] N. Shafii, S. H. Kasaei, and L. S. Lopes. Learning to grasp familiar objects using object view recognition and template matching. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2895–2900, 2016.

[51] Ju Xu and Zhanxing Zhu. Reinforced continual learning. `http://arxiv.org/abs/1805.12369`, 2018.

[52] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. In *International Conference on Learning Representations*, 2016.

[53] Sayna Ebrahimi, Franziska Meier, Roberto Calandra, Trevor Darrell, and Marcus Rohrbach. Adversarial continual learning. `https://arxiv.org/abs/2003.09553`, 2020.

[54] Xu Ji, Joao Henriques, Tinne Tuytelaars, and Andrea Vedaldi. Automatic recall machines: Internal replay, continual learning and the brain. `https://arxiv.org/abs/2006.12323`, 2020.