university of groningen

faculty of science and engineering

Lifelong 3D Object Recognition and Grasp Synthesis Using Dual Memory Recurrent Self-Organization Networks

Krishnakumar Santhakumar

university of groningen

faculty of science and engineering

University of Groningen

Lifelong 3D Object Recognition and Grasp Synthesis using Dual Memory Recurrent Self-Organization Networks

MASTER'S THESIS

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of

Prof. Dr. Hamidreza Kasaei
and
Prof. Dr. Marco Wiering

Krishnakumar Santhakumar (s4035992)

August 31, 2021

Abstract

Humans learn to recognize and manipulate new objects in lifelong settings without forgetting the previously gained knowledge under non-stationary and sequential conditions. In autonomous systems, the agents also need to mitigate similar behavior to continually learn the new object categories and adapt to new environments. In most conventional deep neural networks this is not possible due to the problem of catastrophic forgetting, where the newly gained knowledge overwrites existing representations. Most state-of-the-art models excel either in recognizing the objects or in grasp prediction, while both tasks use visual input. The combined architecture to tackle both tasks is very limited. In this thesis, we proposed a hybrid model architecture consists of a dynamically growing dual-memory recurrent neural network (GDM) and an autoencoder to tackle both object recognition and grasping simultaneously. The GDM part is designed to recognize the object in both instances and categories levels, and the autoencoder network is responsible to extract a compact representation for a given object, which serves as input for the GDM learning, and is responsible to predict pixel-wise antipodal grasp configurations. We address the problem of catastrophic forgetting using the intrinsic memory replay, where the episodic memory periodically replays the neural activation trajectories in the absence of external sensory information. We generate a synthetic dataset to evaluate the proposed model since there is no standard sequential point cloud dataset that can be used in incremental learning scenarios exists. Experiment results demonstrated that the proposed model can learn both object representation and grasping simultaneously in continual learning scenarios.

Acknowledgments

I would like to express my sincere gratitude to my internal supervisor Dr. Hamidreza Kasaei, for his guidance and support throughout this thesis work. The ideas and feedback provided by him during our meetings help to build and improve this research. I also would like to express my special thanks to my second supervisor, Dr. Marco Wiering for devoting his valuable time and providing valuable feedback during this thesis work.

I would like to thank the University of Groningen and the center of Information Technology for giving me access to use Peregrine cluster.

Finally, I would like to thank my parents and friends for their long-standing and unconditional support, without them this journey would not have been possible.

Contents

List of Figures

List of Tables

1 Introduction

Continual Learning is one of the key elements which enables autonomous agents to learn in lifelong learning settings. The problem of lifelong learning is the long-standing challenge in robotics, machine learning, and neural networks [6][7]. One of the major problems that need to be addressed in continual learning is catastrophic forgetting which is caused due to periodic decrease of stability-plasticity dilemma [8]. The tendency to learn a new task without forgetting the previously learned tasks is a stability-plasticity dilemma. The real-world environment is dynamic where the robot needs to learn new information's over time. For example, if a robot is developed for household work the robot should have the capability to recognize (the learnt objects) and incrementally learn the new objects which are not seen during training and need to adapt accordingly. The ability to learn new knowledge in the environment while retaining the previously acquired knowledge is referred to as continual or lifelong learning. The conventional deep learning models are trained on the fixed batches of the large dataset for prediction and classification tasks, which is not suitable for continual learning. In continual learning the data becomes progressively over time while exploring different environments, hence the model developed for continual learning tasks should adapt the newly introduced categories. The continual learning model should concurrently learn the Spatio-temporal representation of the newly acquired data and need to perform better classification while retaining the previously gained knowledge [9] [10].

In the real world, all the prior knowledge to interact with the environment and the direct access to the previous experience is restricted [11] for the efficient lifelong learning model. When conventional deep learning models are trained on sequential tasks where samples are progressively available over time, the performance of the previous task is reduced as learning the new task [12] [13]. The catastrophic forgetting of the conventional static network models can be addressed by retraining the network from scratch. But in the scenarios of developmental learning where the agent learns by interacting with the environment, retraining needs to happen whenever a new observation is observed, which is inefficient and computationally complex. Also, storing all the encountered data is a general drawback in lifelong learning settings. So, the lifelong learning system should be able to learn the new sensory information without forgetting the previously learned task and it should be computationally less complex and with reduced storage complexity.

The human brain and non-human brain have the ability to continually learn knowledge without catastrophic forgetting. Due to neurosynaptic plasticity in the mammalian and non-mammalian brain, yields the physical changes in the neural network which allows to learn, remember, and adapt to the dynamic changes in the environment [14] based on the sensorimotor experiences [15]. Recollecting the separate episodic events, and learning the statistical structure of the episodic events are the two complementary tasks that the brain must constantly perform. The complementary learning systems (CLS) theory [16] provides the computational framework basics for modeling memory consolidation and retrieval. CLS system holds the above two interdependent operations (recollecting the separate episodic events, and learning the statistical structure) which are mediated by the interplay providing means for episodic memory and semantic memory [17][16]. A brief review by Parisi et al. [18] provides insights into the continual learning with neural networks with different approaches. Intrinsic replay or pseudo-rehearsal alleviates the catastrophic forgetting, in which the previous memories are revisited without storing all the data samples by the period replay of the previously encountered samples [19]. The majority of the methods in machine learning and deep learning are designed to address the supervised learning on the annotated dataset (static dataset) not scaling up for the incrementally learning in which the training data is progressively available over time. For the lifelong neural network training, a sequential dataset with temporal meaningful relations needs to be used.

1.1 Scope of this thesis

In this research, we propose a hybrid continually learning model for continuous object recognition and grasping. We created a synthetic sequential point cloud dataset using 3D objects from the shapenet dataset [20] for continuous object recognition and grasping. The proposed architecture consists of growing dual-memory (GDM) recurrent self-organization networks (proposed by Parisi et al. [21]) for the lifelong learning of the spatiotemporal representations of the input data and the convolutional neural network (CNN) based autoencoder for feature extraction, and object grasp prediction. The GDM network comprises two recurrent self-organization memories (episodic memory and semantic memory) that dynamically adapt the number of neurons and synapses. The episodic memory learns the representation of the sensory experience in the unsupervised fashion to dynamically adapt the input features distribution at the instances-level. Whereas semantic memory learns the compact representation of the statistical regularities embedded in the episodic memory at the category level. The problem of catastrophic forgetting is addressed by the periodic replay of the neural activity pattern generated in the episodic memory to both the memories using pseudo-rehearsal or intrinsic memory replay. We conducted different experiments for the batch learning scenario and the incremental learning scenario to evaluate the model performance using the proposed synthetic dataset. We evaluated the performance of the continuous object recognition in the incremental learning task by introducing the new instance (NI), new classes (NC), and new instance and classes (NIC). The performance of the grasp synthesis is evaluated using the intersection of union (IOU) accuracy and grasp success rate on the unseen data. Figure 3.1 shows the overall system architecture for continuous object recognition and grasp prediction in the continual learning settings. The real-time performance of the model towards continuous object recognition and grasping is evaluated by conducting simulated robot experiments.

1.2 Research Questions

This thesis work primarily focus on answering the following research objectives:

1.2.1 How does the sequential point cloud representation of a 3D objects helps to learn the object recognition and grasp affordance in continual learning fashion?

In order to address the simultaneous object recognition, and grasping in continual learning settings, we developed a synthetic sequential point cloud dataset. Here, we developed a dataset generation architecture to generate the sequential point cloud representation of the 3D objects based on different object views and its motion captured by the RGB-D camera. We also employed different augmentation techniques to create different input representations. For dataset generation, we used 3D objects from shapenet dataset [20], and the gazebo library.

1.2.2 Is the point cloud representation of the input data helps the GDM algorithm to learn, and classify the object categories? How does the modified GDM learning helps to improve the performance?

Here we used the generated sequential point cloud samples to train the GDM algorithm. The features extracted from the proposed autoencoder are used as an input to the GDM learning. We conducted experiments on batch and incremental learning scenarios to evaluate the model performance in predicting the instance and object categories. Also, we conducted experiments on different continuous

object recognition scenarios. The influence of the knowledge-based controlled neurons removal and the choice of similarity measure will also be investigated during this study.

1.2.3 Does the proposed learning architecture continually learn the object categories, and perform simultaneous grasping in real-world scenario?

We conducted simulated robot experiments to check whether the proposed model can simultaneously recognize the objects and perform object grasping. The experiments include pick and place, and pack scenarios were the model need to learn the new object categories in the environment to perform the intended tasks. For these experiments, the incremental GDM model with memory replay will be used to incrementally learn, and perform object recognition.

1.3 Thesis Outline

The remainder of this thesis report is structured in five chapters. In chapter 2, the theoretical background related to our work will be discussed. In this chapter literature's related to continual lifelong machine learning, object recognition, and object grasp synthesis prediction is summarized. In chapter 3, our approach towards synthetic sequential dataset generation is explained. Also, the detailed explanations of our proposed model architectures are given in chapter 3. The experimental setup and the results obtained for different learning approaches are discussed in chapter 4. This chapter also has the real-time evaluation of the model performance in a simulated environment for continuous object recognition and grasping. In chapter 5, the brief discussion about the conclusion derived from the different experiments are presented. The potential shortcomings of our proposed approach and the possible future work to improve the model performance are also discussed.

2 Theoretical Background

In this chapter, we give a brief introduction to the theoretical concepts related to this thesis work. In addition to the theoretical concepts, we also provided with the review about the literature works related to continual learning, 3D object recognition, and grasping.

2.1 Convolutional Neural Network

Convolutional neural networks (CNN) [22] is an artificial neural network similar to multi-layer perceptrons [23] (regular neural network), which takes an input and predicts the output by a series of mathematical operations by learning internal weight representation of the neurons in the network. The MLP receives a one-dimensional input vector which is not suited for handling high dimensional input representation such as images because images are made up of a 3D matrix of height H, width W, and channel C (usually refer as depth). When the images of bigger size (for example, $200 \times 200 \times 3$) are transformed to a one-dimensional vector, the regular neural network (MLP) suffers from the addition of more neurons and hidden layers (fully connected layers) thus increase in more training parameters which would lead to over-fitting issues. CNN addresses this issue by the introduction of ConvNets, a ConvNet is usually made up of convolution layers, pooling layers, and fully connected (FC) layers. It also consists of different types of layers like flatten layer, normalization layer, and de-convolution (or transpose convolution) layers, where each one preforms specific operations. Figure 2.1 shows the example CNN architecture for the handwriting digits classification task.

2.1.1 Convolution and Transpose Convolution Layers

The convolution layer (CONV layer) is one of the most important layers in CNN, which mainly makes CNN's differ from other neural networks. The convolution layer is used as feature extractors for the given input image through convolution operation. Each convolution layer in the CNN consists of a set of the learnable parameter named filters, depending upon the type of filters learned during training

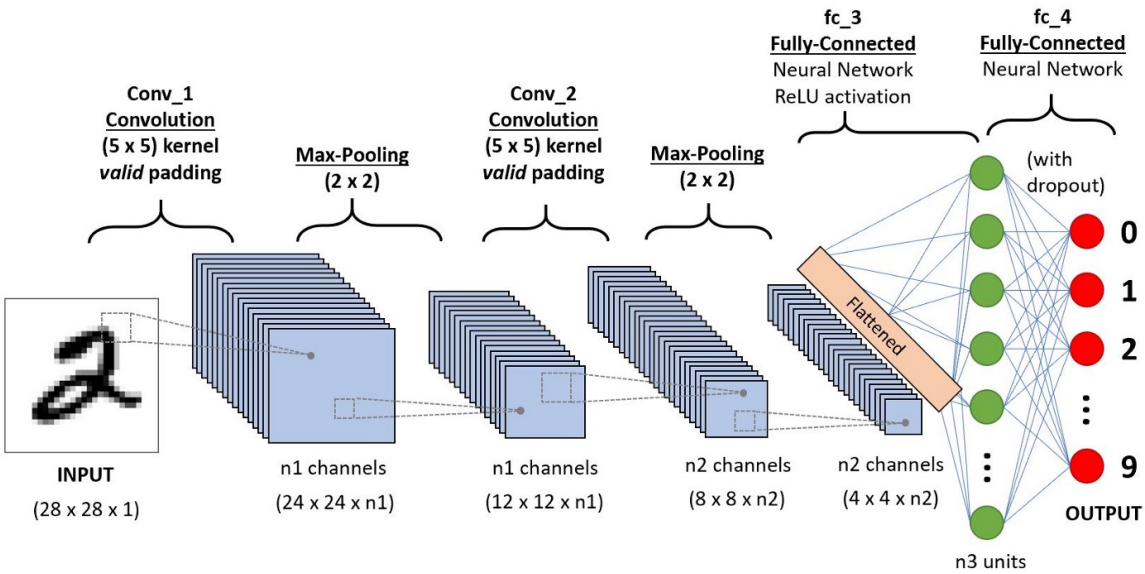


Figure 2.1: Application of convolutional neural network in handwriting digits classification from article [1].

the features like edge detection can be inferred from the input image. The convolution operation is a dot product between the entries of the filter and the input image pixel during the forward pass of the network which produces a 2-dimensional activation map. Based on the type of requirements during convolution operation, the computed values will pass to certain non-linear activation functions, for example to remove negative values in the extracted features. The convolution layer is usually considered as a way of compressing the input data representation into different features, whereas transposed convolution or de-convolution layer decompresses the learned feature map into images. The transpose convolution operation upsamples the learned feature map into desired output feature map. The transpose convolution operations are mostly used in semantic segmentation techniques.

2.1.2 Pooling Layers

Pooling layer is used to reduce the spatial size of the activation maps from the convolution layer. The pooling layers are mostly used for dimension reduction to reduce the learning parameter, computation time and to prevent over-fitting. The pooling layers do not have trainable parameters. The most common types of pooling operations used in CNN are max-pooling and average pooling. In our work, we used average pooling for the dimension reduction on the object representation for GDM learning. In average pooling, the feature maps are downsampled by calculating the average among the values of fixed window size (filters) and strides.

2.1.3 Activation Function

The activation function in the CNN layer defines the transformation of the weighted sum of inputs to the desired outputs. The activation function is applied at the output of the convolution operation in CNN, the fully connected layers and in the output layer. The activation functions can be divided into two groups, linear activation, and non-linear activation function. In this thesis work, we used rectified linear unit (ReLU) to remove the negatives values in the features of grasp map prediction. The function of ReLU for input x is as follows,

$$ReLU(x) = max(0,x) \tag{1}$$

while the values from the convolution operation is passed to ReLU activation, only the positive values are retained and the negative values are replaced with zero based on the formula 1.

2.1.4 Loss Function

The loss function is used to estimate the error values between the predicted output values with the ground truth labels. The goal of the model training is to reduce the error value estimate from the loss function. Based on the loss value, the internal weights of the network are updated using backpropagation. The process continues until the end of certain learning epochs or stop criteria is attained. In this work, we used the smooth L1 loss function known as Huber loss, which is less sensitive to outliers and handles the problem of exploding gradients. The smooth L1 loss function is given as,

$$L(x,y) = \begin{cases} 0.5(x-y)^2, & \text{if } |x-y| < 1, \\ |x-y| - 0.5, & \text{otherwise.} \end{cases} \tag{2}$$

Here, x is the actual (or ground truth) value, and y is the predicted value. In smooth L1 loss function, if the absolute error falls below 1 the squared difference is calculated otherwise an absolute representation of x, and y is calculated. The problem of large numbers computed by squared difference

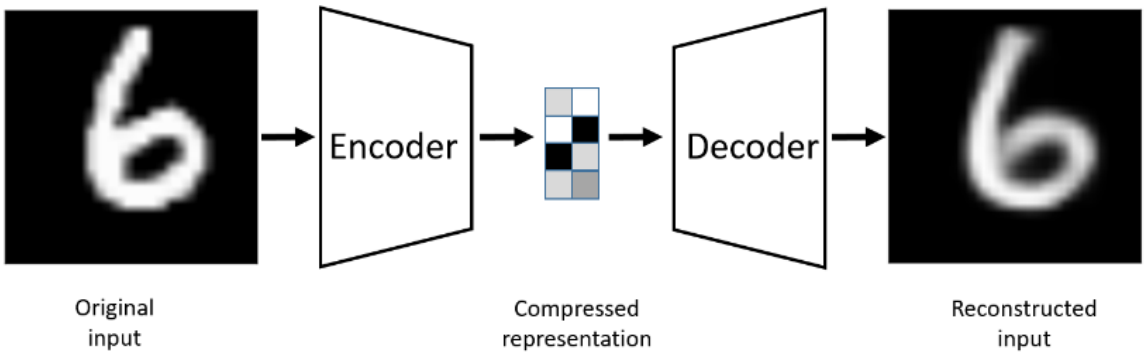


Figure 2.2: An autoencoder example [2].

values is avoided in smooth L1 loss, as for different values of x and y greater than 1 the values are not squared.

2.2 Autoencoder

Autoencoder [2] is a set of artificial unsupervised neural networks where the inputs and outputs are the same. An autoencoder takes data x as an input and predicts x' as an output. The autoencoder consists of encoder part (with encoder function, $h = f(x)$) which learn the meaningful feature representations of the input data x and a decoder part (with decoder function, $r = g(h)$) which reconstructs the input (similar as possible to x) with the learned feature representation $x\prime$. Depending on the type of input data representation, the encoder and decoder are made up of regular networks (ANNs), convolutional neural networks (CNNs), or other types. Figure 2.2 shows the example of autoencoder application in handwritten digit reconstruction.

The performance of the autoencoder is evaluated by measuring the reconstruction loss, which measures the difference between the original input and the reconstructed output. The autoencoder model is trained to minimize the reconstruction loss by using backpropagation to maximize the network predictions performance.

2.3 Grow-When-Required Neural Network

A grow-when-required (GWR) neural network is an unsupervised machine learning technique proposed by Marsaland et.al., [3]. Unsupervised machine learning techniques are often used for clustering and dimension reduction from high dimensional input space to low dimensional feature space (K-means clustering, principal component analysis (PCA)). The unsupervised ML techniques like Kohonen Self-Organizing Map (SOM) can be used for classification tasks [24]. In SOM and other similar networks, the network structure and dimension need to be fixed prior to the learning and the capacity of the networks (i.e., the number of nodes) is also fixed. Also, in SOM networks the number of associated neighbors is fixed, which means that the network can only able to adapt the existing connections hence it cannot be able to add new ones. This makes SOM like networks not suitable for learning the non-stationary or sequential data or continual learning. To address these issues, GWR network was proposed. GWR network can dynamically grow or remove the neurons in the network based on the activity of the neurons in the network for the given input data distributions. Similar to the neurons in other neural networks, the GWR network has neurons its weights and the edges to connect

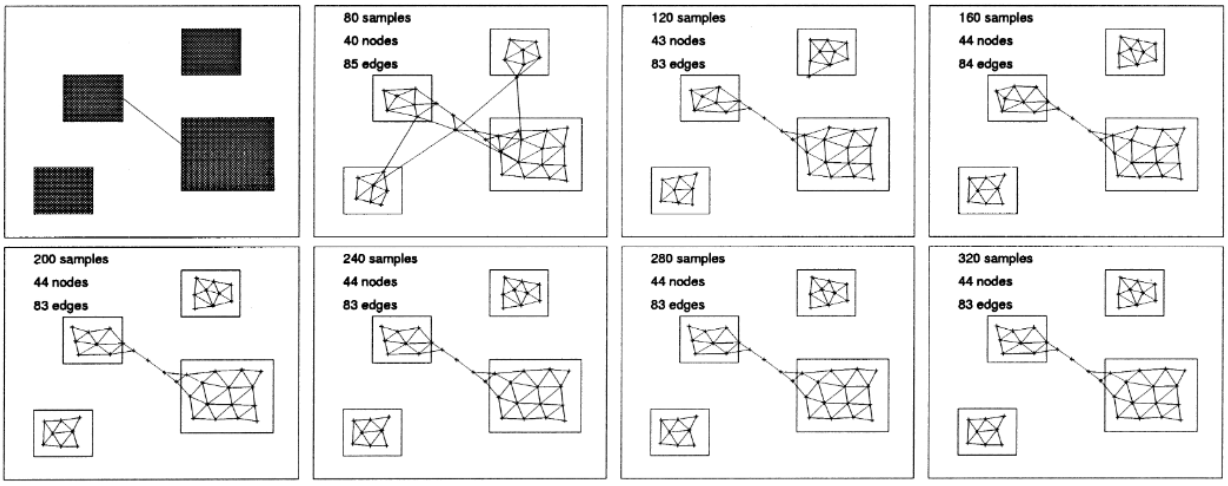


Figure 2.3: An Grow-When-Required (GWR) network example. Image taken from [3]. This example image illustrates the learning of square boxes by GWR network.

with neighboring neurons. In the GWR network both the edges (i.e., connections) and neurons can be deleted during the learning process. The neurons in the GWR network is recurrent which means it can have more than one neighbors based on the similarities between the nodes in the network. In the GWR network the connections between the nodes are created based on the Best Matching Unit (BMU) prediction (i.e., between first BMU and second BMU) for the input data.

GWR uses distances based similarity measures (e.g., Euclidean distance) to compute the BMU pair. The GWR network adds new nodes whenever the activity of the BMU neuron is low. The activity of the neuron is a function of the distance between the weights of the node and the input. When the new neurons are created, the weight of the newly created neurons are determined based on the average of the best matching nodes and the input data. This can be seen as a genetic algorithm, where offsprings are created based on the shared knowledge from the parents (here parents refer to BMU neuron and input data). Unlike other unsupervised techniques where the winner takes all the credits, in the GWR network the winner and its associated neighbors also get updated (at different learning rates). In this way, not only the best matching unit represents the input data adapts its associate nodes with similar knowledge get adapted. This way of updating the BMU and its neighbors, the data with spatial structural representations can be identified and classified, which means that GWR can handle the non-stationary sequential input data. The detailed functions of the GWR network and its algorithm are described in the paper proposed by Marsaland et.al., [3]. Figure 2.3 shows the example of the GWR network in learning the black square boxes in the input. Each block in the figure 2.3 (except the first one) indicates the learning step of the GWR network. It is be noted that GWR network starts its learning process using only two neurons and gets added and updated during the learning process. From the blocks in figure 2.3, we can see that the GWR network creates, destroys, and updates the connections and neurons when the input sample size is increased. We can also observe that after certain learning steps, specifically after the third block (in figure 2.3) the number of nodes and connections are constant, this indicates that the GWR network is regularized for the input data and no other additional neurons or connections are required to represent the input. From the above interpretation, we can see that GWR networks are less prone to over-fitting, despite the increase in a number of sample size over learning with constant neurons and edges (or connections) GWR algorithm addresses the problem of over-fitting.

2.4 Gaussian Noise and Down Sampling

Gaussian noise or Gaussian probability distribution is a statistical noise which has the probability density function (PDF) equal to normal distribution [25]. For a Gaussian random variable x the probability density function p is given as follows,

$$p_G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3}$$

where μ is the mean, and σ is the standard deviation. Adding Gaussian noise to point cloud data helps the learning algorithm to be robust against the external noise in real-time applications. Point cloud down-sampling is the way to reduce the number of points in the point cloud data which represents the 3D objects by using voxel grids [26]. In each voxel, all the points presented are downsampled based on their centroid. Down-sampling a point cloud in the dataset ensure the model to recognize the 3D objects even with the small number of point cloud points, thus makes the robust prediction performance.

2.5 Entropy

Entropy is a measure of randomness or uncertainty in the information being processed [27]. It expresses the probabilistic and heterogeneity or impurity relationship in the information. The mathematical expression of entropy p is expressed in equation 4.

$$Entropy(p) = -\sum_{i}^{N} p_i log_2 p_i \tag{4}$$

where p_i is the probability of $i, i \in 1,..,N$.

2.6 Literature Review

This study addresses the problem of continual learning of 3D object recognition and grasping in open-ended domains. In this section, we reviewed the most relevant literature's which inspires this research work. The following sections consist of brief reviews on continual learning, object recognition, and grasp synthesis.

2.6.1 Continual Learning

Several works have been published to address continual learning using different techniques, such as complementary learning systems (CLS) [17], regularization methods [28, 29, 30], dynamical architectures [31, 3, 32, 33, 34], and memory replay techniques [35]. We refer the reader to a brief review on continual learning in deep network by Parisi et al. [18]. It provides a good insights into the continual learning using different approaches.

In general, the problem of continual learning is usually addressed by several regularization methods. The elastic weight consolidation (EWC) [28] addresses continual learning in supervised and reinforcement learning (RL) scenarios by mitigating the catastrophic forgetting. Zenke et al., [29] proposed an approach to alleviate the catastrophic forgetting by allowing the individual synapses in the learning model to estimate their importance in the learned task. An ensemble method, named PathNet [30], uses a genetic algorithm to find the optimal path through a neural network of fixed size to find which parts of the neural network can be reused for learning new task while freezing

task-relevant parts is developed to avoid catastrophic forgetting. Although such regularization methods provide a way to alleviate the problem of catastrophic forgetting, they are limited by the number of neural resources for learning new tasks over time which may lead to the performance trade-off between the old task and the new task.

To address the limitation of neural representations in the regularization methods, several dynamical architectures were proposed. For instance, Dynamically Expanding Network (DEN) [31] increases the trainable parameters while learning the new task using network expansion and selective retraining by sparse regularization in a supervised learning paradigm. Such networks learn the representation of new tasks by dynamically increasing the network size. This way, the problem of limited neural resource is addressed. Recently, S. Jain & H. Kasaei [36] reported state-of-the-art results in open-ended 3D object recognition using pre-trained convolutional neural network (CNN) and DEN. Similar to DEN, the combination of the self-organizing incremental neural network and a pre-trained CNN, proposed by Part and Lemon [33, 37], allows the network to grow in a continuous object recognition scenarios. Although the dynamically expandable networks adapt to learn a new task, it needs to have access to the entire dataset while learning the new task which increases the storage complexity.

The continual learning models for robotics application have to address the problem of complex data storage. Towards this goal, Marsland et al., [3] proposed a neural network, which grows when required (GWR) based on the synaptic neural activity triggered by the input data distribution to the best matching similarity nodes in the network, and it also deals with the dynamic data distributions. An extended version of the GWR model, called Gamma-GWR [34], embeds the gamma memory [38] during the neuron growth for learning the short-term temporal relation representations of the input data distribution in the absence of external sensor information. This extended version of GWR network address the problem of data storage in continual learning by learning the short-term temporal relation. Using Gamma-GWR, Parisi et al., [34] showed the state-of-the-art results in batch learning scenarios with missing and corrupted sample labels.

Apart from the above-mentioned methods, the concept of dual-memory systems were developed to address short-term and long-term memory consolidation. The system in which each synaptic connection has two weights: the plastic weights (to preserve long-term knowledge) and the fast-changing weights (which holds the temporal short-term knowledge). One of such dual-memory system was proposed by Gepperth and Karaoguz [32], using modified self-organizing maps (SOM) and SOM extended with short-term memory (STM) to address the incremental learning task by alleviating the catastrophic forgetting. In their work, they used STM to store the previously learned knowledge and replayed back while learning the new task. Even though SOM + STM address continual learning it also has certain limitations. Since the STM has the limited capacity it overwrites the old task while learning a new task and it also requires storing the entire dataset during incremental training.

The above-mentioned methods designed for the classification of the static data representations in supervise learning paradigm. In more natural settings, the data representations are sequential, where the underlying spatio-temporal relations are incrementally available over time (i.e. objects information with different features representations). For the continual learning, a sequential dataset with temporal meaningful relations needs to be used.

The growing dual-memory (GDM) network architecture [21] is better at representing the spatio-temporal relation of the input data in the lifelong settings with reduced storage and computational complexity. Furthermore, the GDM network achieved state-of-the-art result in continuous object recognition scenarios. The GDM network consists of deep transfer learning-based pre-trained CNN, followed by the two different recurrent self-organizing gamma grow when required networks (Gamma-GWR), named episodic memory (learns the sensory experience) and the semantic memory (learns the task-relevant signals). The GDM memories can dynamically adapt the number of neurons and

synapses based on the input data distribution. Using the intrinsic memory replay or pseudo-rehearsal the problem of catastrophic forgetting is alleviated, in which the previous memories are revisited without storing all the data samples by the period replay of the previously learned temporal synapses (during training) [19]. In our approach, we utilized the GDM networks to continually learn the objects categories in an open-ended scenario and, at the same time, we enhanced its prediction performance by introducing the regulated neuron removal based on the neuron activity and its knowledge about the input data. Moreover, We employed different similarity measures to obtain the optimal performance with the input data to estimated the best matching unit (BMU) in the GDM networks.

2.6.2 Object Recognition

The continual learning approaches reviewed in the previous section 2.6.1 mostly provide solutions to object recognition in machine learning. In this section, we discussed some of the research works related to 3D object recognition. The instance-based method proposed by Luca et.al., [39] address the object recognition in open-world domains by using the memory-based incremental framework, which stores each and any object that encounter in the environment to learn discrimination between known and unknown objects. Since this methods stores, each objects instances encountered it is not feasible for long-horizon continuous object recognition tasks [40, 41]. The another approach named OrthographicNet proposed by Kasaei et.al., [42], an instance-based object recognition approach which interactively learns based on Teach, Ask, and Correct functions. The OrthographicNet can learn new object categories from very few onsite examples, which is clearly an advantage when compared to the memory-based incremental learning framework. In recent work by Ayoobi and Kasaei et.al., [43], their interactive open-ended 3D object recognition learning using Local Hierarchical Dirichlet Process (Local-HDP) [44] shows the state-of-the-art performance in terms of memory efficiency, accuracy, and scalability.

Similar to interactive learning, view-based approaches are also published to address the 3D object recognition. The multi-level CNN proposed by Sambit et.al., [45] learns the multi-scale feature representations to increase the 3D object recognition accuracy. This method learns the feature representation from 3D spatial geometry representation on objects like point clouds, 3D models, and RGB-D data. This type of multi-level learning addresses the limitations of structure feature representations (voxel grids, octress) and the limitations of unstructured feature representations (graphs and point clouds), such as restrictions on resolution, tree depth, and challenges due to non-uniformity in the data samples. The another approach based on volumetric and multi-view CNNs is proposed by Charles et.al., [46] to address the object classification on the 3D objects. This approach utilizes two distinct volumetric CNN architectures and multi-view CNN with multi-resolution 3D filtering. In-addition authors proposed auxiliary tasks to predict the class labels for the object classification. In all the above-mentioned methods for 3D objects recognition, the neural network layers are fixed and trained on fixed batches of a dataset. This means to learn the new task, the output layer size needs to be increased and in most cases additionally, the neurons in the fully connected layers (prior to the output layer) need to be added to learn the new task. In most cases to learn a new task (object), the models need to be trained with including all the data samples of previously learned tasks (objects), which is computationally expensive and has a complex storage problem. In continual learning of 3D object recognition the models needs to address the computational complexity and storage problem.

2.6.3 Grasp synthesis

Several deep learning based algorithms have been proposed for object grasping recently. For examples, Morrison et al., proposed a generative grasping convolutional neural network (GG-CNN) [47] that can predict pixel-wise grasp configuration for never-seen-before objects. The empirical and analytical approaches of different grasp synthesis techniques are given in the review by Sahbani et.al., [48]. Mahler et.al., [49] proposed an dexterity network (Dex-Net) to predict the robust grasp planning using Multi-Armed Bandit model with correlated rewards. A vision-based robotic grasping system was proposed by Jincheng Yu et.al., [50] to tackle both 3D object recognition and pose estimation using Max-pooling convolutional neural network (MPCNN). Muti-modal geometric learning approach proposed by Watkins-valss et.al., [51] uses the depth and tactical information to create rich and accurate 3D models for robotic manipulation tasks. Then they used 3D CNN to learn the object grasping using the depth and tactical information from their developed 3D models. Interactive open-ended learning approach to recognize the multiple objects and their grasp affordances for robotic manipulations was proposed by Hamidreza Kasaei et.al. [52, 53]. This system uses verbal and kinesthetic teaching to learn the grasp affordance category and the grasp configurations and it uses Bayesian based approaches for learning and recognizing the object categories.

Another approach named Res-U-Net [54], used encoder-decoder based CNN architecture to predict objects' grasp affordances first, and then used a search policy to find the best path to approach and grasp the target object. In another work, Kasaei et al., [55] proposed a method to address multi-view 3D object grasping based on convolutional auto-encoder. Kumra et al., [56] developed a generative residual convolutional neural network (GR-CovNet) which generates the antipodal grasp for the given n-channel input. All the above reviewed methods are good at predicting the grasp synthesis for the given 3D objects but they are not able to recognize the object category label simultaneously. To address the simultaneous object category recognition and grasping, Kasaei et al.,[57] coupled a generative mixed auto-encoder with a probabilistic 3D object recognition approach to simultaneously predict the pixel-wise grasp configuration and recognize the objects in open-ended domains. The model receives the input from multiple views of the given 3D object, and then, simultaneously predicts the pixel-wise grasp synthesis and a compact representation for an active object recognition task. In particular, the authors proposed an active learning strategy to teach, ask, or correct the prediction of the model while learning the object categories in an open-ended scenario. Unlike our work, these approaches followed a single-shot prediction and completely discard the spatio-temporal information.

3 Methodology

In this chapter, we describe our approach towards simultaneous object recognition and grasping in a continual lifelong learning fashion. Our approach to lifelong learning of object recognition, and grasping comprises three major tasks. In task 1, we generated a synthetic sequential point cloud dataset for continuous object recognition and grasp prediction learning. Since there is no existing dataset representing the sequential relation of 3D objects are available, we created our synthetic dataset. In task 2, we proposed an autoencoder model to extract the low dimensional (256 dimensions) feature vector for object recognition and to estimate the pixel-wise antipodal grasp points. The antipodal grasp points are estimated based on the grasp map generated in the form of quality, width, and angle images predicted by the autoencoder network. Task 3 is to develop the recurrent GDM networks consists of episodic and semantic memory to predict the instance and category level accuracy in the continual learning fashion with controlled neuron growth. The overall system architecture for the continuous object recognition and grasping is shown in figure 3.1, which comprises the proposed autoencoder, GDM networks, and a simulated teacher to teach the neural network when new object categories are encountered in real-world fashion.

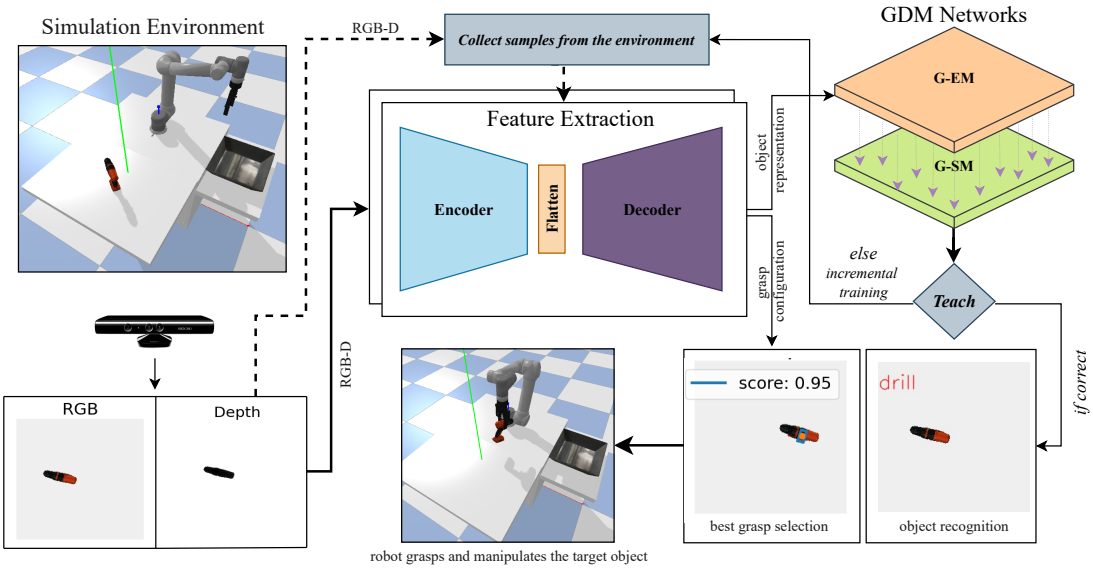


Figure 3.1: Overall system architecture for continuous object recognition and grasping. RGB-D data representing the 3D object in the environment is fed to generative autoencoder to obtain, *(i)* the object representation for object recognition using GDM networks and *(ii)* grasp configuration for object grasping. If the new object category or instance is introduced to the robot, based on the teacher input the system automatically collect the samples from the environment and its associated label information to continual learn and adapt to the new sensory information.

3.1 Dataset Generation

The continual learning model needs to be trained on the sequential dataset, where the spatiotemporal relations of the task are progressively available over time, similar to natural world settings (e.g dataset comprises of videos). Real-world objects are three-dimensional, the dataset like Cornell, Jacquard [58], ModelNet [59] and ShapeNet [20] which comprises of 3D objects of different categories are used for object recognition, and grasp synthesis prediction. Similar to the above-mentioned dataset lot of dataset are available to represent the 3D objects for different tasks. But all the dataset provides the static representation of the objects. To address lifelong learning, we need a dataset with the

sequential representation of the 3D objects. CORe50 dataset [60], a dataset developed to address the continuous object recognition problem, since the dataset is made of 2D images it cannot be used for grasp synthesis prediction. So, we developed our custom dataset which represents the sequential point cloud of the 3D objects in the form of different scenes. Where each scene consists of objects that belong to different instances and categories, similar to the representation of the scene in the CORe50 dataset. The point cloud samples are described in the form of 3D coordinates ([x, y, z]) and RGB information. The dataset generation is constructed in a way that binary representation of the point cloud samples can also be generated. A point cloud sample of an object is represented by a set of points p_i, where $i \in [1, 2, ..., n]$.

Table 3.1: Objects used in the dataset generation

Cooking pan	Bowl	Car	Toy pistol	Pen
Mug	Knife	Chair	Cereal Box	Pencil
Guitar	Bottle - water	Clock	Hammer	USB
Soda can	Airplane	Headphone	Staple	Spoon
Bottle - wine, beer	Jug	Vase	Eraser	Fork

We developed the dataset using the robot operating system (ROS), and 3D objects sampled from the ShapeNet dataset [20], and gazebo environment. In our dataset generation process, we used 125 object instances belongs to 25 categories in the representation of 15 scenes. Each object instances in the scene contain 500 samples of point cloud information of the respective objects in a sequential manner. Table 3.1 comprises the list of objects used in the dataset generation. The objects included in the CORe50 dataset are distinct from one another, but in a real-world scenario, the objects belong to different categories with identical features exists. For example kitchen utensils like spoons and fork which may have identical structure but used for a different purpose. Learning and classifying these kinds of objects is hard. Hence, we included objects like pencil and pen, spoon, and fork in the dataset which belong to different categories but similar in shape. The main objective of including this complexity is to check whether our proposed model performs well at classifying these fine-grained details. Since we used only 125 objects to create 15 scenes, the object instances from one scene are used to generate the dataset samples for another scene. For example, object from scene 1 is used in scene 6 and scene 11, objects from scene 2 is used in scene 7 and 12 and follows. Even though we used the objects from one scene to create another scene, we make sure that samples generated differ from one another. To create more diversity between the object samples in each scene, and to make the model prediction robust to external noise, we augmented the dataset. The dataset augmentation includes changing the position (x-axis, y-axis, and z-axis), and orientation (roll, pitch, and yaw) of the object, adding Gaussian noise to the point cloud data, down-sampling the point cloud, adding random occlusion, and adding Gaussian noise along with point cloud down-sampling.

3.1.1 Architecture for Dataset Generation

The dataset generation architecture pipeline is shown in figure 3.2. The system initially spawns the 3D objects to the gazebo world using the ROS spawn model service, based on the object scale value, and the scene number. With our dataset generation algorithm, the object of any size can be used in the dataset generation by the custom scaling feature to fit the object to the gazebo environment. The scale values are determined prior to the dataset generation by experimenting with the different scale values to fit the 3D object into the gazebo world. Then the object region of interest (ROI), and its associated

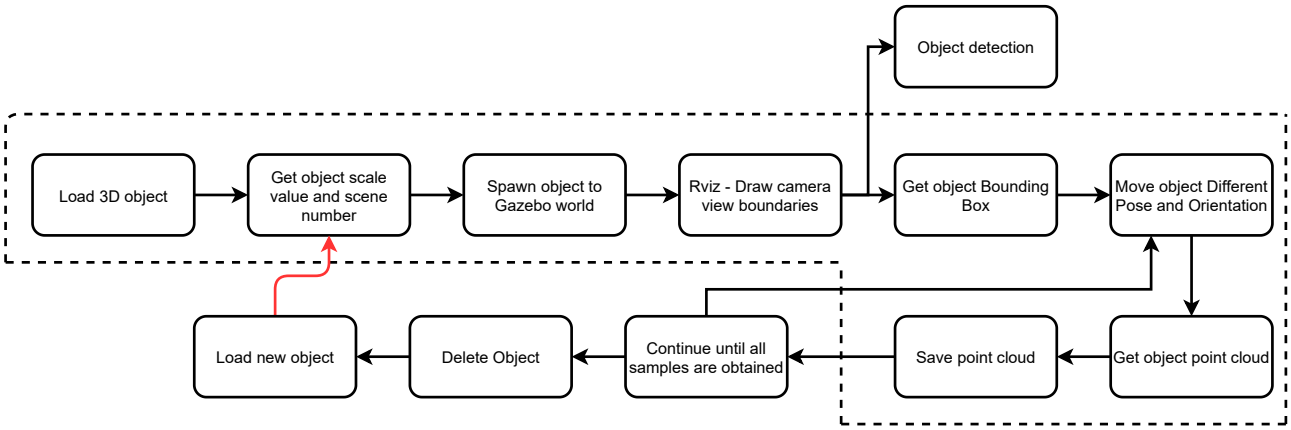


Figure 3.2: Architecture pipeline for the dataset generation.

bounding box are detected. After which the object is moved to a different position and orientation (at the beginning the position and orientation will be zero). For the current position, and orientation of the given 3D object, the point cloud data is generated. The point cloud with binary or RGB information can be generated based on the user requirement during the point cloud generation process. In the process along with the change in position and orientation, the above-mentioned augmented techniques are introduced. The data sampling continues until the maximum frame (here 500 frames) is obtained. Once the required point cloud samples are generated and saved, the current object will be removed from the gazebo world and a new object will be spawned. This process continues until all the object instances of 25 categories in all the scenes are generated.

3.2 Model

The proposed learning model architecture consists of two main elements, the deep transfer learning-based autoencoder for feature extraction and grasping, and the dual-memory recurrent self-organization networks to predict the instances and category level labels for the given input data. The autoencoder used in our approach deals with the prediction of low-dimensional feature vectors for object recognition, and the grasp map (G_i) for object grasping. The low-dimensional feature vector predicted by the autoencoder servers as an input for the GDM networks, for object category learning. Figure 3.3 shows the proposed model architecture for lifelong object recognition and grasp prediction.

Initially, the system receives the multi-view representation of the sequential 3D point cloud data for the given object at different frames at time t. Then the obtained point cloud is converted into the RGB-D image of width W and height H, then it is used as an input $X_t^i \in \mathbb{R}^{W \times H}$ to the model learning. The RGB-D data with time-dependent sequential input is passed to the autoencoder to get the 256 dimensional feature vector. The Gamma-GWR uses distance-based similarity measures to compute best matching units (BMUs). To eliminate the discrimination caused by high dimensional, and spare data representation, we performed a series of convolution operations to reduce the dimension of the feature vector to a size of 256. Then the 256 dimensional feature vector is used as an input to the growing dual memory recurrent self-organizing networks (GDM) to predict and classify the object categories. In parallel to the object category prediction, the sequential RGB-D point cloud data are fed to the autoencoder to predict the pixel-wise grasp configuration map (G_i) in the form of angle (Θ_i), width (w_i), and quality (q_i) image representations ($G_i = (\Theta_i, w_i, q_i) \in \mathbb{R}^{H \times W}$). Initially, the sequential data is processed to obtain the optimal grasp view by the entropy selection for object grasping, if time t is greater than 1. If the sequential data contains only one frame (i.e., at the time $t = 1$), the optimal

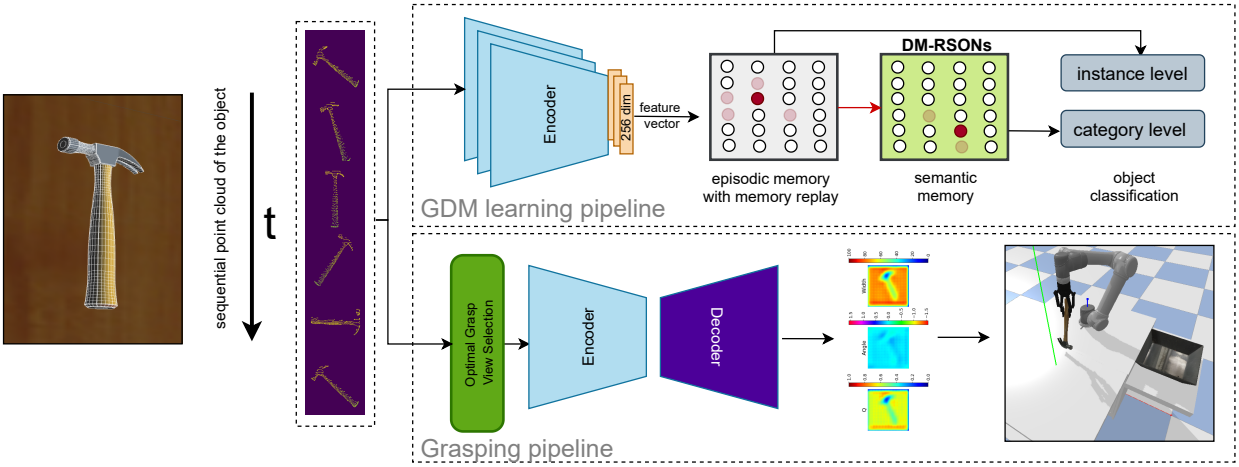


Figure 3.3: Proposed model architecture: for continual object recognition and grasp synthesis learning using mixed autoencoder, and growing dual-memory network (GDM). The sequential point cloud samples from the input 3D object are initially generated and converted to an RGB-D image, then fed to the network to obtain instance and category level object representation and pixel-wise grasp configuration. For object recognition, all the sequential samples are being used and for the grasping, the object view with maximum entropy is used.

grasp view will be equal to the input point cloud object view. Based on the quality, width, and angle images, the grasp rectangles are estimated which are then converted to robot space to perform object grasping and manipulation.

3.2.1 Autoencoder

To extract the feature vectors for object recognition learning, and to predict the object grasping, we developed a convolutional neural network (CNN) based autoencoder model. In the autoencoder model, we employed a deep transfer learning approach by using the pre-trained generative residual convolutional neural network (GR-ConvNet) [56]. Since GR-ConvNet is the current state-of-the-art model in grasp prediction learning it suits well to our task. The encoder output of GR-ConvNet is a high dimension of size 401408 ($56 \times 56 \times 128$), which is not suitable for GDM learning. To reduce the dimension of the features vector, we developed an autoencoder architecture with GR-ConvNet as a pre-trained model. Figure 3.4, shows the proposed autoencoder architecture.

The encoder part receives an n-channel input RGB-D image of size 224×224, and is fed to the fine-tuned GR-ConvNet encoder to get the features matrix of $56 \times 56 \times 128$ dimensions. Then followed by the two 2D convolution layers of size $31 \times 31, 64$ filters and $5 \times 5, 32$ filters with batch normalization and rectifier linear unit (ReLU) as an activation function. The second convolution layer of the custom encoder consists of two identical branches, one used for the growing dual-memory (GDM) learning and another one for grasp synthesis. For GDM learning, the output of the second convolution layer in the custom encoder (without ReLU activation) is fed to the 2D average pooling layer followed by the flatten layer to obtain the feature vector of 256 dimensions. For the grasp synthesis, the output from the second convolutional layer with ReLU activation is passed to the custom decoder for grasp prediction learning. The custom decoder of the architecture consists of two 2D deconvolution (transpose-convolution) layers of size $15 \times 15, 32$ filters and $31 \times 31, 64$ filters with batch normalization and ReLU activation function, followed by the fine-tuned GR-ConvNet decoder. The output layer of the decoder has three images of 224×224 dimensions to predict the quality (Q), angle ($\cos(2\Theta) + \sin(2\Theta)$), and width (W) of the grasp points. The predicted grasp quality (q_i) score value ranges from 0 to 1. The quality score 0 indicates a lower confidence value to predict the grasp, whereas a score

of 1 indicates the higher confidence in success to predict the grasp. The amount of angular rotation (Θ_i) required to grasp the object of interest at each point ranges $[-\frac{\pi}{2}, \frac{\pi}{2}]$. The required width w_i of the gripper which is expressed as a measured depth, and has the value in the range of $[0, W_{max}]$ (W_{max}, is the maximum opening of the gripper width expressed in pixels). Using the predicted quality, angle, and width images the grasp point and locations are calculated by estimating the local peaks in the quality of the image predicted. The grasp rectangles are determined based on the estimated grasp points location (based on the number of grasps required), and the predicted angle and width images. To execute the obtained grasp in the robot, the predicted grasp in image space needs to be converted to robot space by using the cascade transformation as a function of G_r [56].

$$G_r = T_{rc}(T_c i(G_i)) \tag{5}$$

where, the transformation T_{ci} converts the image space to the camera space by using the camera's intrinsic parameters, and the transform T_{rc} converts the grasp points (G_i) in image space to robot space.

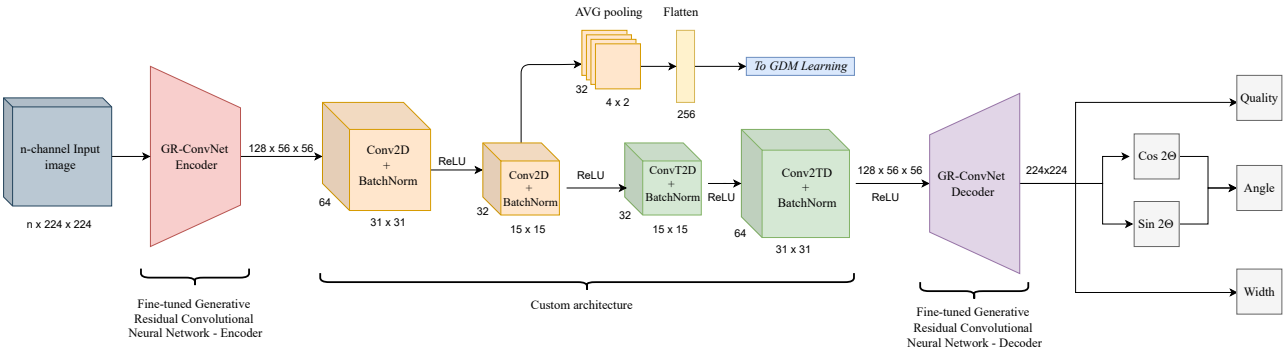


Figure 3.4: Proposed autoencoder architecture: *(i)* To extract feature vector for object recognition learning, and *(ii)* to predict the quality, angle, and width images (which then used to estimate the grasp points), for the given n-dimensional RGB-D image input.

We fine-tuned the layers of pre-trained GR-ConvNet to adapt to the newly added convolution layers. The learning accuracies are estimated with smooth L1 loss function ($\mathscr{L}(G_i, \hat{G}_i)$), and the model is trained using Adam optimizer [61]. As a result, the episodic memory of the dual-memory network receives the input feature vector, $x^i(t) \in \mathbb{R}^n$ of size 256. The episodic memory learns metrics of features relations in the unsupervised fashion. The discrepancy between the sequential input and neural representation is minimized by creating new neurons or updating the existing neurons based on the activation threshold, habituation rate of the neurons, and other hyper-parameters (explained in section 4.3). Then the learned weights from the episodic memory W_b^{EM} (based on the total number of neurons) are passed to the semantic memory to learn the task-specific knowledge. Thereby episodic memory results in the prediction of instance-level information and the semantic memory with the category level prediction.

3.2.2 Growing Dual Memory Networks (GDM)

The object prediction and classification is performed by the unsupervised based learning self-organizing growing dual-memory networks (GDM) [21]. The 256 dimensional features values predicted by the autoencoder model are used as an input for the GDM learning. GDM network consists of episodic, and semantic memory for instance, and category level object prediction based on the sequential representation of the input data. Both the episodic and semantic memory uses the gamma Grow-When-Required (Gamma-GWR) [34] network, which dynamically grows or shrinks based on the input data

distribution. The neural network structure of the Gamma-GWR is recurrent, where the connection between the neurons is determined based on the similarity measure (e.g. Euclidean distance) estimated between the input data distribution, and the best matching unit (BMUs) in the neural network. In which neurons may have more than one neighboring relation based on the similarity between the sensory information received from the input data. Also, the gamma memory in the Gamma-GWR holds the temporal relation of the neural activation trajectories during learning which gets dynamically changed relative to the input data distribution. This temporal memory is used during pseudo-rehearsal or intrinsic memory replay to alleviate the catastrophic forgetting in the incremental learning tasks. The networks are initialized with two neurons and it dynamically grows by holding the spatiotemporal relation while iterating over the input data samples. Each neurons A, in the network consist of weight vectors $w_j \in \mathbb{R}^n$ and K context descriptors $C_{k,j} \in \mathbb{R}^n$. For each given input $x^i(t) \in \mathbb{R}^n$, the best matching unit (b) is calculated based on the Manhattan distance metrics (given in equation 6 - 8). In the work by Parisi et al., [21] the authors used euclidean distance as a similarity measure. But in our work, we used Manhattan distance to measure the similarity in the input data distribution. We analyzed the performance of the prediction using different distance-based similarity measures (Euclidean distance [62], Squared Euclidean distance [63], Manhattan distance [64], Minkowski distance [65] with the power of 3, Mahalanobis distance [66], and cosine similarity [67] measures), based on experimental results we found that Manhattan distance metric suits best for our chosen input data distribution.

$$d_j = \alpha_0 |x_i(t) - w_j| + \sum_{k=1}^{K} \alpha_k |C_k(t) - c_{j,k}| \tag{6}$$

$$C_k(t) = \beta . w_b^{t-1} + (1-\beta) . c_{b,k-1}^{t-1} \tag{7}$$

$$b = arg \min_{j \in A} (d_j) \tag{8}$$

where α and β are the constants that regulate the temporal context influences. w_b^{t-1} is the weight vector of the BMU at time t-1. $C_k \in \mathbb{R}^n$ is the global context descriptors with $C_k(t_0) = 0$, and $c_{b,k-1}^{t-1}$, the context descriptor of the BMU, and k-1 descriptor at time t-1. The neurons in each network are either created or existing neurons are updated based on the activity of the neuron a(t), and its habituation counter h_j, regulated by the insertion threshold (a_T) and habituation threshold h_T. The activity of neuron a(t) is determined based on the distance relationship between the input and BMU (b) which is computed as follows,

$$a(t) = \exp(-d_b) \tag{9}$$

when the neuron with its respective BMU predicts the input sequence correctly, that respective neuron results in the highest activation value of 1. Each neuron is equipped with the habituation counter h_j. The habituation counter ($h_j \in [0,1]$) expresses the frequency of the neuron firing in the training process. The habituation values of the BMU (b) and it neighbor (n) decreases as the frequency of the neuron firing increase. The habituation rule [3] for a neuron i (equation 10) is given by,

$$\Delta h_i = \tau_i . \kappa . (1 - h_i) - \tau_i \tag{10}$$

τ_i and κ are the constants that control the monotonically decreasing behavior of the habituation counter, the habituation counter of BMU (h_b) decreases faster than neighboring neurons (h_n). Compared to other conventional unsupervised learning algorithms where the winner takes all the credits, in GWR not only the winner but also its associated neighboring neurons are also gets updated. The weight vectors and the context descriptors are get updated where ever the new neurons are created and

existing nodes are updated. When new neurons are created its weights are computed as the average weights of BMU, and input. The weight, and context descriptor update for the neuron i is given as,

$$\Delta w_i = \varepsilon_i . h_i (x(t) - w_i) \tag{11}$$

$$\Delta c_{i,k} = \varepsilon_i . h_i (C_k(t) - c_{i,k}) \tag{12}$$

where ε_i is a learning rate, the learning rate of BMU (ε_b) will be higher than the learning rate of neuron (ε_n). The connection between the two neurons (BMU and second BMU) is created when two neurons fire together. Each neuron that exists in the network has a certain age, when those ages reached a certain threshold it will be removed from the network. The ages between the first BMU and second BMU rest to zero, whereas other neighboring ages are increased by the value of 1. At the end of training epochs, the neurons with an age higher than the threshold, and the neurons which do not have neighbors are removed from the network. The associative matrix $H(j,l)$, stores the sample labels based on the frequency of input distribution during the learning phase. For each neuron j, the model stores the number of times that an input with label l has j as its BMU. Then the predicted label (ξ_j) for the neuron j can be computed as,

$$\xi_j = arg \max_{l \in L} (H(j,l)) \tag{13}$$

where l is the arbitrary class label. When the existing BMU neuron (b) is updated. The associative label matrix (H) is updated by, $\Delta H(b,\xi) = \delta^+$ and $\Delta H(b,l) = \delta^-$, where δ^+ and δ^- are labelling constants. When the new neuron (r) is added and the label to the input sample x(t) is ξ, the associative matrix (H) increased by one row and initialized according to $H(r,\xi) = 1$ and $H(r,l) = 0$ with $l \in L \setminus \xi$, where L is a set of class labels. If input sample label ξ is not present in L, a new column is created in H and then initialized to $H(b,\xi) = 1$ and $H(b,l) = 0$. If the input label to the sample $x(t)$ is not known or not labelled, then H is not updated. Therefore, based on the associate matrix labelling without the pre-defined number of class labels, the unsupervised Gamma-GWR can be used for the classification. Algorithm representation of the Gamma-GWR network for the episodic and semantic memory is given in algorithm 1.

3.2.3 Episodic Memory

In episodic memory (G-EM), the neuron's growth is unsupervised. Based on the insertion threshold (a_T), and the habituation threshold (h_T), the network learns a fine-grained representation of the input data. The new neurons in the network will be created only when the activity of the neuron $a(t)$ falls below a_T (i.e., $a(t) < a_T$, equation 9). The temporal connection of the neural activation trajectories are learned during episodic memory training by sequence selective synaptic links [21]. When two neurons are activated continuously their temporal synaptic link P(i,j) are increased by $\Delta P_{(i,j)} = 1$. For each neuron i∈A, the next neuron v of a prototype vector can be retrieved by selecting $v = arg \max_{j \in A/i} P(i,j)$, where i and j represent the neurons at time t-1 and t. The temporal representation learned in episodic memory are used during memory replay in the absence of external sensory input to retrieve the previously gained knowledge while learning new ones. During the learning phase, the neurons in G-EM store the instance class labels (ξ^I) and the category label l^C. The associative matrix $H(j,l^I,l^C)$ (equation 13) stores the instance, and category label for each neuron j.

3.2.4 Semantic Memory

Similar to episodic memory, semantic memory (G-SM) is also associated with the Gamma-GWR network. Instead of receiving the direct sensory information as input, G-SM received the episodic

Algorithm 1: Gamma-GWR network [34]

1. Initialize the network with a set of two random neurons, A=w_1, w_2 with empty context vectors C_i^k for $k = 1,2,...K$ and $i = 1,2$;
2. Initialize the K empty global context descriptors C_k=0;
3. Initialize empty set of connections $E = \emptyset$;
4. Initialize empty temporal matrix, $temporal[num\_neurons, num\_neurons]$=$\emptyset$;
5. Compute alphas for a = K+1 context descriptors, α_a=$\frac{\alpha_a}{\sum_a \alpha_a}$; $\alpha_a = exp(-a)$, a=1,2,...K+1;
6. Initialize the hyperparameters;
7. At each epoch, iterate through input samples $x(t)$ with labels ξ;
8. Update the global context descriptor at time t, $C_k(t)$ by equation 7;
9. Find the first (b) and second (n) BMU neurons by equation 6, b=$arg\min_{i \in A} d_i(t)$, n=$arg\min_{i \in A \setminus \{b\}} d_i(t)$;
10. Compute the BMU neuron activity $a(t)$ by equation 9;
11. **if** $a(t) < a_T$ *and* $h_b < h_T$ **then**
    - 11.1 add a neuron r, $(A = A \cup \{r\})$: $w_r = 0.5(x(t) + w_b)$, $c_r^k = 0.5(C_k(t) + c_b^k)$;
    - 11.2 update labels by equation 13;
    - 11.2 update edges between neurons: $E = E \cup \{(r,b),(r,n)\}$ and $E = E \setminus \{(b,n)\}$;
    - 11.3 update temporal synapses of n: $\Delta P_{(i-1,n)} = 1$ and expand temporal size for new neuron;
    - 11.4 for new neuron r add the habituation firing counter value 1;

    else
    - 11.5 reduce the firing counters of BMU neuron (b) by habituation rule by equation 10;
    - 11.6 update label value of b : $\Delta H(b,\xi) = \delta^+$, $\Delta H(b,l) = \delta^-$, with $l \in L \setminus \xi$;
    - 11.7 update weights and context descriptors of winning neurons and its neighbors by equation 11 and equation 12;
    - 11.8 update edges between neurons: $E = E \cup \{(b,n),(n,b)\}$;
    - 11.9 update temporal synapses of b: $\Delta P_{(i-1,b)} = 1$ 11.10 reduce the firing counters of BMU (b) neighbors by equation 10;

    end
12. Increment the ages of all the edges connected to BMU (b) by 1;
13. At the end of each epoch remove all the old edges by algorithm 2;
14. Repeat step $7-13$ for all the epochs;
15. At the end of all epochs remove neurons without edges (i.e., connections);

weights w_b^{EM} as input, based on the number of neurons in the G-EM and the associative matrix containing the label l^C information. Unlike G-EM, G-SM creates new neurons only when the labels predicted by the BMU are miss-classified with the ground truth labels. Since G-SM uses category-level signals to regulate the network growth, the same neurons may be activated for the different instances of the object which belongs to the same category. Compared to G-EM neurons, G-SM neurons will activate for the higher temporal windows as a result of hierarchical processing. Since G-SM neurons receive input from the G-EM network which fires for $K^{EM} + 1$ of the input frames, results in each neuron of the G-SM to code for a total of $k^{SM} + K^{EM} + 1$ input frames. K^{EM} and K^{SM} are the number of context descriptors in the episodic, and semantic memory. This result G-SM in a high degree of temporal slowness than G-EM, which yields G-SM to learn more compact representation of semantically related input from the episodic memory experience.

3.2.5 Memory Replay

Memory replay is addressed as the representation of pseudo-patterns generated in terms of temporally ordered neural activation trajectories based on the temporal synaptic information learned during the G-EM training, using pseudo-rehearsal or intrinsic memory replay [19]. The pseudo patterns are generated in terms of recursively reactivate sequence-selective neural activity trajectories (RNATs) [21] embedded in the temporal synapses of G-EM. The neural trajectories are computed for each neuron in the episodic memory at the fixed temporal window, and replayed back to the G-EM and G-SM after each learning episode triggered by external input. RNATs computation doesn't explicitly need storing the temporal relation, and the labels of the previously seen samples to remember the past knowledge. Since, it generates the sequence-selective prototype sequence during each learning iteration which will be periodically replayed back to G-EM and G-SM networks during the learning phase. The RNATs S_j, for each neuron j in the episodic memory for the length of $\lambda = K^{EM} + K^{SM} + 1$ are computed as follows,

$$S_j = (W^{EM}_{s(0)}, W^{EM}_{s(1)}, .., W^{EM}_{s(\lambda)}) \tag{14}$$

$$s(i) = arg \max_{n \in A \setminus j} P_{(n, s(i-1))}, i \in [1, \lambda] \tag{15}$$

where $s(0) = j$, and $P(i, j)$ is the temporal synapses define by $arg \max_{j \in A \setminus i} P(i, j)$. The class labels of the pseudo-patterns in s_j can be retrieved using equation 13.

3.2.6 Controlled Neuron Growth

Algorithm 2: Controlled edges removal

```
Input: n_i ∈N, Number of neurons in the network
Output: n_i ∈N, Neurons with old connections removed
for i = 1,..,N do
    for j in neighbors of n_i do
        if ages[i, j] > max_age then
            if h_i > N_T then
                rest connections[i, j] to zero;
                rest connections[j, i] to zero;
                age[i, j] to zero;
                age[j, i] to zero;
            else
                age[i, j] to zero;
                age[j, i] to zero;
            end
        end
    end
end
```

In the work by Parisi et al., [21], at the end of each epoch the connection between the neighboring neurons are removed when the age of a particular node reached its maximum age. At the end of the training, the neurons without neighbors are removed. When a neuron is removed from the network, the associated information gain of that particular neuron about the input data distribution is also

completely removed, independent of its knowledge about the data. In the scenario when a neuron learns useful information at the beginning of training for a particular object category that has few neighbor connections, and it is not triggered further. The age of that neuron gets increased as the training process continues, at the end of training due to its high age value that neuron is removed despite its knowledge about the input data. In this paper, we addressed this problem by using the regulated neuron connection removal. The connection between the neurons is removed when its habituation value $h(t)$ (which holds the knowledge level of the neuron), is greater than the removal threshold value (N_T). If the neuron habituation value is greater than the removal threshold value (N_T), and its age is higher than the maximum age, the age of the neuron and its connection between the associated neighbors are reset to 0. If the habituation value of the neuron is less than the threshold value (N_T), and it has a higher age value, the connection between its neighbors are retained only their age values are rested to 0. In this way, we can retain the neurons which have a better understanding of the input data. From the experimental observations, we observed that the controlled neuron removal significantly improves the prediction performance in episodic and semantic memory. The pseudo-code of this method is shown in algorithm 2.

3.3 Data sampling and Pre-processing for GDM learning

We used our proposed sequential point cloud dataset for GDM learning (as explained in section 3.1). The GDM model is trained on two learning scenarios batch learning and incremental learning. In these learning strategies, the representation of the input data sequence is completely different. For batch learning, the model needs to have access to the entire data sample comprising of all the objects from all the scenes. But for incremental learning, the samples that belong to different object categories (in our case 25) need to be available in the form of mini-batches. In incremental learning training, the mini-batch created will be equal to the number of object categories in the dataset. This form of data representation helps the model to learn the new sensory information in an incremental manner, thereby addressing continual learning. We also designed the data sampling procedure in a way that a small subset of sequential samples from the full dataset can be retrieved, so that the model can learn with the desired number of samples instead of the need to use the entire 187500 samples. The GDM network receives the feature vector representation of the input point cloud data predicted by the autoencoder. Hence, all the sequential samples are fed to the trained autoencoder model to get their respective feature vectors. Since GDM is based on distance-based similarity measures during pre-processing all the feature vector values are normalized.

Algorithm 3, shows the pseudo-code of the data sampling technique. As a result, data for bath learning will be of the shape (sample size $\times$256), and for incremental learning it will be of the shape (number of categories $\times$ sample size $\times$ 256). If the mini-batches are sampled, both batch, and incremental learning have batch size b_s before its data structure (i.e., the shape of batch learning data changed to ($b_s\times$ sample size $\times$256), and the shape of incremental learning data changed to ($b_s\times$ number of categories $\times$ sample size $\times$ 256)).

Algorithm 3: Data sampling

```
Input: Dataset representation in from of Scenes (S) format (S1 - S15)
Output: Sampled data 𝒟, Instance label e_labels, Category labels s_labels
initialize empty e_labels array ;
initialize empty s_labels array ;
for s = 1,...,15 do ;                                   /* Number of scenes */

    for c = 1,...,25 do ;        /* Total number of object instances per scene */

        for d_{s,c} = 1,...,500 do ;        /* Total number of samples in each object
         instance in each scene */

            predict feature vector for d_{s,c} ;          /* using trained autoencoder */
            normalize the feature vector;
            add to data 𝒟;
            append instance label of d_{s,c} to e_labels;
            append category label of d_{s,c} to s_labels;
        end
    end
end
if learning type == batch learning then
    if mini batch is True then ;            /* subsets of data from full dataset */

        split 𝒟, e_labels, s_labels based on the batch size b_s;
        modify 𝒟, e_labels, and s_labels;
    else
        𝒟 = 𝒟;
        e_labels = e_labels;
        s_labels = e_labels;
    end
end
if learning type == incremental learning then
    split the sampled 𝒟, e_labels, and S_labels based on the total number of object category C ;
     /* Here, 25 */
    modify 𝒟, e_labels, and s_labels;
    if mini batch is True then
        split the modified 𝒟, e_labels, and s_labels based on batch size b_s;
        update 𝒟, e_labels, and s_labels;
    end
end
return 𝒟, e_labels, and s_labels;
```

4 Experiments and Results

The experiments and performance evaluation of the proposed system consists of four phases, *(i)* the generation of synthetic sequential point cloud dataset, *(ii)* feature extraction and grasp prediction, *(iii)* the performance evaluation of object recognition and classification of the GDM learning and *(iv)* the evaluation of the proposed system in the simulated robot environment.

In the first set of experiment phases, we describe the experimental setup, the training procedure, the choice of data augmentation parameters, and the results obtained for the dataset generation. Secondly, we present the result of the autoencoder model on the Cornell dataset [68] and the results of feature extraction and grasp map prediction on the generated point cloud dataset, where we also compared and discussed our results with the GR-ConvNet [56] model. Then the object recognition and classification performance of the GDM learning is evaluated in the third phase. The GDM learning is evaluated in batch and incremental learning scenarios, where we also evaluate the performance of the incremental learning model in the continuous object recognition scenarios, new instance (NI), new class (NI), and new instance and class (NIC). The training procedures and choice of hyperparameters of the GDM learning scenarios are also explained in this section. We also present and discussed the influence of controlled neuron removal and their results during the GDM learning at the episodic and semantic memory level. Finally, we performed the real-time evaluation of the proposed system architecture (see fig. 3.1) on the simulated robot in the context of *pick and place*, and *pack* scenarios. During the simulated robot experiments, the efficiency of the model in incrementally learning the new object categories while retaining the previously learned knowledge is tested and evaluated.

4.1 Dataset

As we mentioned, we generated our own synthetic sequential point cloud dataset for continuous object recognition. We used a robot operating system (ROS) framework and gazebo environment to create the dataset. We utilized 3D objects from the shapenet [20] library and gazebo repository to create the dataset. The dataset comprises 125 objects instances from 25 categories in the form of sequential point cloud samples. The dataset also embedded with different augmentation conditions such as a change in translation and position of the object, adding Gaussian noise to the point cloud data, down-sampling the point cloud data, and adding occlusions to the 3D objects. The complete dataset generation pipeline and the techniques used to generate the dataset are explained in section 3.1.

4.1.1 Environmental setup

The environmental setup for the dataset generation is shown in figure 4.1. The dataset generation environment consists of the RGB-D Kinect camera to capture the RGB, depth, and point cloud representation of the 3D objects in the vision of sight. The object shown in table 3.1 are used in dataset generation. Initially, for all the objects the scale size (i.e., object size to fit the gazebo world) is determined by using the custom scaling features in the dataset generation. Figure 4.2, shows the sample object instances from each category in table 3.1 in the gazebo environment where all the objects are re-scaled to fit the inside the wooden floor. Then the re-scaled 3D objects are assigned to their respective scale and scene folders. For example, the headphone object to be placed in the s1 scene folder with the scale size of 0.5 has the directory structure of "$\backslash s1 \backslash 0.5 \backslash headphone$". Note that the 3D objects used in the dataset generation should need to have COLLADA (i.e., *.dae*) extension, the 3D

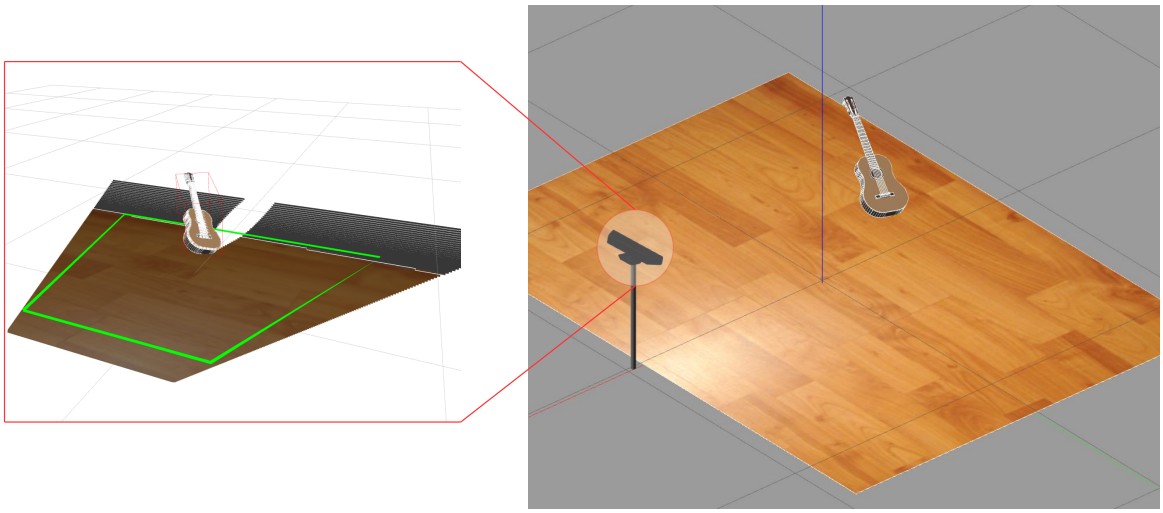


Figure 4.1: Dataset generation environmental setup: Right - Gazebo environment with Kinect camera, wodden floor, and the guitar object (a 3D object); Left - the kinect camera view in Rviz, a RGB point cloud view of the guitar object with detected bounding box (shown in red color around the object) and the green line representing the boundary markers for the object movement.

Figure 4.2: Sample object instances from 25 categories used in the dataset generation.

objects from the other format are not supported. For each data sample created, the respective file is encoded with its scene number, the model index within the scene, roll, pitch and yaw (i.e., object orientation), and the frame count, which is then later used during the data sampling procedure to retrieve the class name and for other purposes. The training procedure for the dataset generation is given in algorithm 4.

4.1.2 Parameter Selection

The parameters and its range for different augmentation techniques used in dataset generation are as follows: in gazebo environment the position of the x-axis ranges from $0.176m$ to $-0.7m$, the position of the y-axis ranges from $0.072m$ to $-0.2475m$, and the position of the z-axis is set to a constant value of $0.003076m$. The position of the z-axis value is set to constant to prevent the object's elevation from

Algorithm 4: Dataset Generation procedure

1. Initialize the ROS service to spawn, delete, set and get model states and bounding box services ;
2. Initialize the dataset generation and augmentation parameters (section 4.1.2);
3. Iterate over the scene folders $(s1 - s15)$;
4. **for** *models in scene folder* **do**
    5. add 3D object to the gazebo world at the origin based on the scale value;
    6. set frame count to zero $(t = 0)$;
    7. draw bounding box markers;
    8. change position and rotation of the object ;
    9. get the bounding box of the object at current position and orientation;
    10. get the current state of the 3D object in the gazebo world;
    11. update the objects position in the gazeo world ;
    12. **if** *scene folder in* $(s1 - s5)$ **then**
        12.1 read and save the point cloud sample of the object at frame t;
    end
    13. **if** $scenefolder > s5$ **then**
        14. **if** *number of frame is range* $15\% - 49\%$ **then**
            add Gaussian noise to point cloud data;
        end
        15. **if** *number of frame is range* $50\% - 74\%$ **then**
            add down sampling to point cloud data;
        end
        16. **if** *number of frame is range* $75\% - 100\%$ **then**
            add Gaussian noise with down sampling to point cloud data;
        end
        17. **if** *number of frame is range* $0\% - 14\%$ **then**
            do nothing;
        end
        18. save the point cloud sample;
        ; `/* Note the` μ, σ `and ds percentage will be changed randomly for each iteration. */`
    end
    19. Increase the frame count by 1;
    20. repeat step $8 - 19$ until the maximum number of frames $(i.e., 499)$ is obtained ;
    21. If maximum number of frames obtained delete the current 3D object from the model;
end
22. Repeat step 3, until the data is generated for all the objects in all the scenes;

the wooden floor. For a change of orientation, the roll value ranges from 0 degree to 360 degree with the offset of 90 degree, the pitch value ranges from 0 degree to 360 degree with the offset of 60 degree and the yaw rotation value ranges from 0 degree to 720 degree with the offset of 30 degree. For the Gaussian noise, the mean (μ) value changed between 0.01 to 0.51 with the difference of 0.01 and standard deviation (σ) values changed between 0.005 to 0.011 with the difference of .001. Both μ and σ values ranges are determined based on experimentation and both of these values are randomly sampled during the dataset generation process. For down-sampling, the point cloud data obtained

from the Kinect camera are down-sampled based on voxel size ranges from 0.01 to 0.1. The point samples are down-sampled from 1% to 10% of the total size of the respective objects point cloud. The percentage ranges are determined to ensure that even though the point cloud data points are down-sampled, the overall structure of objects needs to be retained. We inducted this constrain to have the point cloud data with a considerable amount of points to extract the meaningful features for object recognition and grasping.

4.1.3 Results

The data samples generated using the different augmentation techniques are shown in figure 4.3 for car and jug objects. For each object, the samples are generated using the Gaussian noise, down-sampling, Gaussian noise with down-sampling, and occlusion augmentation. From the results of the dataset generated we can observe that despite the use of different augmentation techniques, the overall structure of the object is retained. At the end of dataset generation processes for 15 scenes, a total of 187500 samples are generated at the rate of 2.5 frames-per-second (fps). Also, to note that all the samples in the generated dataset are sequential, where the sample at time $t-1$ holds spatiotemporal relation with the sample at time t. Hence, the generated dataset is well suited for continual learning tasks to learn the input distribution in an incremental manner where each data point of the object in the scenes holds temporal relations with one another. The entire dataset generation time was approximately 4 days, 4 hours, and 45 minutes. The dataset size at the end of the dataset generation process is 4.3 GB.

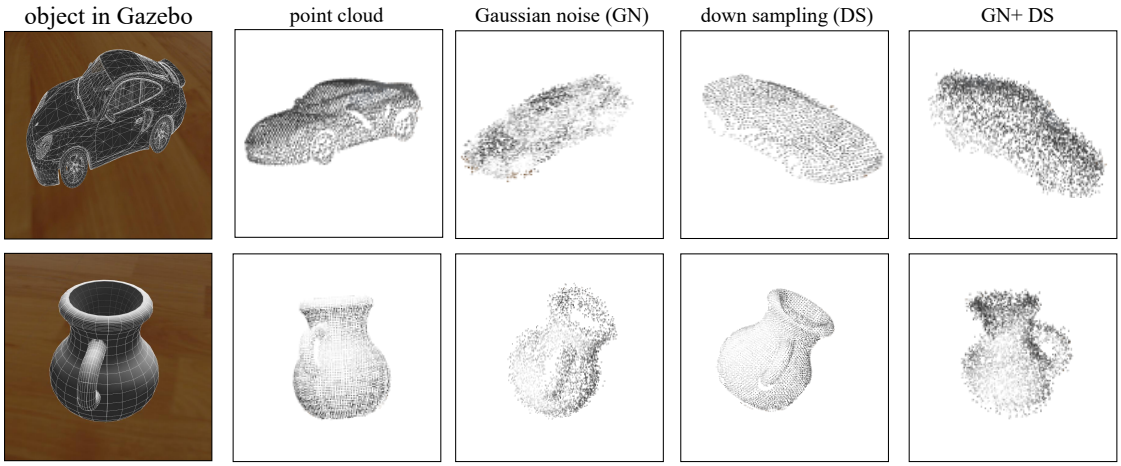


Figure 4.3: Data augmentation: car (*top-row*), and jug (*bottom-row*) in the Gazebo environment followed by their point clouds. For augmenting the data, we have applied: (*i*) Gaussian noise, (*ii*) down-sampling, and (*iii*) Gaussian noise plus down-sampling to the point cloud of the object.

4.2 Object Representation and Grasp prediction

We trained the autoencoder model (figure 3.4) using the Cornell dataset [68], the dataset which is used in the GR-ConvNet [56] training. The dataset is augmented by using techniques like including random rotation and zoom. We used 90% of the dataset for training and 10% for testing. The network receives input of 4-channel 224×224 RGB-D images, so all the data points are converted into RGB-D images and resized to 224×224 before being fed to the network. The network weights of the GR-ConvNet are initialized from the pre-trained model on the Cornell dataset and it is fine-tuned during

the training process. The model is trained with adaptive momentum (adam) [61] optimizer and with a smooth L1 loss function. The smooth L1 loss function is used to handle the problem of exploding gradients during the training. The model performance is evaluated using the intersection of union (IOU) metrics score between the ground-truth and the predicted grasp rectangle. The grasp points detected are valid only when the intersection of union (IOU) score between the ground-truth grasp rectangle and predicted grasp rectangle is more than 25%, and the grasp orientation offset between ground-truth and the predicted grasp rectangle is less than 30 degree, based on the rectangle metric [68]. We trained the autoencoder model for 10, 20, 25, 30, and 35 learning epochs with batch size 4 and evaluated the performance on the test data with batch size 1. As a result, we obtained the best overall IOU accuracy of 93.9% on the test dataset with the model trained for 35 learning epochs. The results are average across five learning trials. The result which we obtained was 2.7% less compared to GR-ConvNet results on the Cornell dataset. This results in an accuracy drop are due to the dimension reduction in the encoder part for the GDM learning. Even though there is an accuracy drop, the model performed better in grasp prediction during our test on the unseen objects.

The autoencoder model is developed and trained using the PyTorch library in python. We trained the autoencoder on the NVIDIA RTX-2070 Max-Q GPU, and the training time takes within a range of two hours for the different training epochs. The total number of trainable parameters of the autoencoder model is $2,262,308M$ million params, where the pre-trained GR-ConvNet model has the total trainable parameters of $1,900,900M$ million, and the custom-designed convolution, and deconvolution (or transpose convolution) layers consist of $361,408k$ trainable parameters.

4.2.1 Grasp prediction on the unseen objects

Similar to the GR-ConvNet model, the IOU match is determined based on the grasp points (by which the grasp rectangles are estimates) identified by the local maximum peaks in the predicted quality (Q) image. From the heat map of autoencoder predicted output (see fig. 4.4) of the width, angle, and quality images, we can observe that local maximum values represent the object and its grasp information better. From the network output, we can also see that the model is good at predicting the object shapes for the point cloud based input. The local maximum points are estimated with the threshold value of 0.25, and with the minimum distance between points as 5 pixels. Similar to the above result, n number of pixel-wise grasp points can be estimated within the object boundaries. Figure 4.4 shows the grasp points estimate based on the local maximum peak detection with two grasp points estimated for the hammer, crackerbox, and scissors objects. The top row shows the results of grasp points estimation for hammer object, the middle row shows the results for crackerbox object, and the results of scissors object are shown in the bottom row. The circles in the grasp points image indicate the grasp rectangle center, the blue rectangle indicates the first grasp rectangle whereas the orange rectangle indicates the second estimated grasp rectangle.

To note that in the Cornell dataset the data points are recorded for the real-world objects whereas in our generated dataset the data points are the samples of simulated objects. Despite the difference, we can see that the model is good at predicting the grasp points for all the given input samples. Since the model is trained on the real-world objects and its good prediction performance on the simulated objects, this model can be used for both real-world and simulated environments to extract the features for GDM learning, and the grasp synthesis prediction. Also, most of the objects used in the dataset generation process are unseen objects or new objects to the trained model. For example, in the scissors object instance in the figure 4.4, we can see that the model is good at the grasp prediction on the unseen objects. For all the input object samples (from the generated dataset, and objects used in

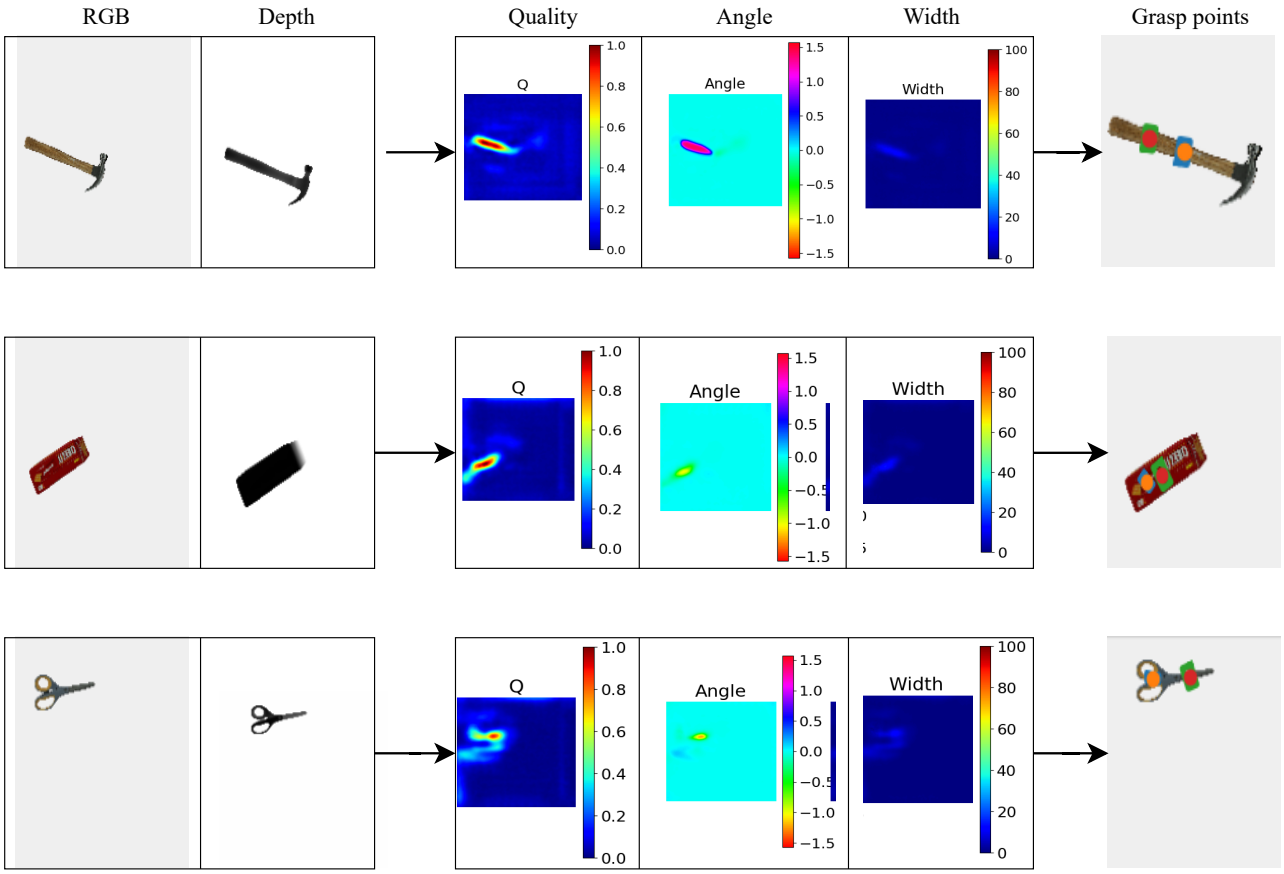


Figure 4.4: Feature extraction and grasp point prediction: (i) the RGB and depth images input, (ii) the three columns indicates the quality, angle and width images predicted by the autoencoder model, and (iii) followed by the local maximum points estimated based on the autoencoder output, and the grasp rectangles determined based on the estimated grasp points, circles indicates the center of the rectangles.

simulated robot testing) that we tested using this model, we observe that the model is good at learning the object geometrical shapes (for example see fig.4.4 quality, angle, and width images). Since the model is good at learning the geometrical shapes, the low-dimensional (256 dimensions) feature vector extracted from the encoder for GDM learning should also hold a good object representation that has a meaningful feature for object recognition learning and in finding the similarities between the object samples. Hence, from the results, we can say that the proposed autoencoder model servers as a good low-dimensional feature extractor and at the same time it is good at grasp synthesis prediction.

4.3 GDM learning

The performance of GDM learning on instance and category level object classification prediction is evaluated on the batch learning and incremental learning scenarios. For GDM learning the 256 dimensional features vectors predicted by the autoencoder model for the generated sequential point cloud dataset serves as an input. In batch learning, the entire dataset represents in a single batch where all the sequential data samples from all the object instances and scenes are used in the learning process. In incremental learning scenarios, the sequential samples belong to a particular category (i.e., one at a time) are progressively available over time and it is not shown again during the training phase, by this way the performance of the model in alleviating the catastrophic forgetting can be evaluated. The model is trained and tested with the generated sequential point cloud dataset (section 4.1.3). The

performance of the GDM model on batch learning is evaluated by three different experiments. First, the model is trained and test with temporal context (TC) by enabling the model to learn the temporal relations while training. Second, the model is trained with temporal context (TC) but during testing on the test dataset, the TC is disabled to check whether the learned temporal relations in the training phase influence the test phase. In the third experiment of batch learning scenario the model is trained and tested without TC, this experiments servers as the baseline for the comparison of the results.

Then, the performance of the GDM model in an incremental learning scenario is evaluated with and without intrinsic memory replay, [21]. In all the incremental learning experiments the temporal context (TC) is enabled. In addition, the incremental learning model is evaluated on the continuous object recognition scenarios, where the new object instances (NI), new object categories (NC), and new object instances and categories (NIC) are exposed to the model over time. All the continuous object recognition experiments are trained and tested with and without intrinsic memory replay. For both incremental and batch learning, the training procedure, the hyperparameter selection, and the results are discussed in this section. In addition to the performance evaluation of the GDM learning, the effect and influence of the proposed controlled neuron growth are also explained in this section, where we discussed the choice neuron removal threshold N_T parameter based on the accuracy and neuron growth in episodic and semantic memory.

4.3.1 Batch Learning

Table 4.1: Hyperparameter settings for G-EM and G-SM networks in batch learning.

Hyperparameter	**Values**
Insertion thresholds	$a_T^{EM} = 0.7, a_T^{SM} = 0.8$
Global context	$\beta = 0.5$
Learning rates	$\varepsilon_b = 0.3, \varepsilon_n = 0.003, \varepsilon_c = 0.001$
Habituation threshold	$h_T = 0.1$
Habituation function	$\tau_b = 0.3, \tau_i = 0.1, \kappa = 1.05$
Edges removal threshold	$N_T = 0.2$
Labelling	$\delta^+ = 1, \delta^- = 0.1$
Context descriptors	$\alpha_1 = 0.63, \alpha_2 = 0.234, \alpha_3 = 0.086$

In batch learning, the dataset consists of all the objects from all the scenes are used for training. The performance of the model is evaluated on the test dataset based on the instance-level and category-level object recognition accuracy. From the different scenes generated (s1 - s15) (section 3.1), the samples from scenes s8, s6, and s14 are used for testing and the samples from the remaining 12 scenes are used for training. To evaluate the importance of the temporal context (TC) on the point cloud data training. Similar to Parisi et al., [21], we experimented the batch learning under three different conditions GDM with TC, GDM without TC during testing (by setting context descriptor k=0, during testing), and GDM without TC (by setting context descriptor k=0, during training and testing). The hyper-parameters used during the batch learning are listed in table 4.1. Note, that these values are determined based on the results of several experiments with different choices of the parameter values, the values are shown in table 4.1 give the best overall results. The number of context

descriptors (K) is set to 2 for both G-EM (K^{EM}) and G-SM (K^{SM}) networks. Based on the selection of the context descriptors, G-EM neurons activate for 3 image frames and G-SM activates for 3 G-EM neurons, which means G-SM activates for processing window of 5 frames. The number of context descriptors is determined based on different experimental observation values. For batch learning, we synthesized 120 samples from each object category from the training dataset (total of 3000 samples) to train the GDM model. To reduce the training time and to evaluate the model performance on the less sample size we reduced the training dataset size, but in testing, we used the entire test dataset with 500 samples from each object category. The training procedure for the batch learning algorithm is shown in algorithm 5.

Algorithm 5: Training procedure for batch learning

1. Initialize the hyperparameters;
2. Get $data$, e_{labels}, and s_{labels} for the batch training using the data sampling algorithm 3;
3. Initialize the G-EM and G-SM networks;
4. Train G-EM network with $data$, e_{labels}, and s_{labels} using the Gamma-GWR algorithm 1 for n epochs;
5. Get the episodic weights, and the episodic labels from the G-EM training;
6. Train the G-SM network with the episodic weights and episodic labels using the Gamma-GWR algorithm 1;
7. Compute the test accuracy of the G-EM and G-SM network;
8. Repeat step 4-7 until the end of training epochs;
9. **if** *no TC during test* **then**
   - 9.1 set the TC to zero (i.e., K=0);
   - 9.2 Compute the final test accuracy of the G-EM and G-SM network;

   else
   - 9.3. Compute the final test accuracy of the G-EM and G-SM network;

   end

Results and Interpretations: The results of the batch learning model on category level during training and testing are shows in table 4.2. The results are obtained after 35 learning epochs which are averaged across five learning trials by randomly shuffling the bathes from different scenes.

From the results (shown in table 4.2), the GDM with temporal context gives the best accuracy compared to the other two approaches. We obtained average accuracy of 75.55% (instance level) and 88.12% (category-level). When tested on the unseen test samples on the test dataset, we obtained an overall average accuracy of 81.67%. Figure 4.5, shows the result of batch learning with TC where the results are averaged across five learning trials. Figure 4.5(a) represents the number of neurons created over each training epoch in episodic (G-EM) and semantic (G-SM) memory, figure 4.5(b) shows the accuracy of instance and category level over the learning epoch, figure 4.5(c) shows the average quantization error at the G-EM and G-SM, and the figure 4.5(d) shows the category-wise object classification accuracy on the test data (s6, s8, and s14), where the final bar groups indicate the average overall classification accuracy of the model on the test data. When comparing the result of the GDM model with TC and the GDM model without TC, the instance level accuracy is improved over 6.96% and the category level accuracy shows 4.44% improvement respectively. Dropping the temporal context of the GDM model during the testing phase shows 2.95% improvement during model validation on training (86.83% - without TC test, and 83.68% - without TC), but gives more or less equal results while testing on the test dataset (79.82% - without TC test, and 80.07% - without

Table 4.2: Classification performance comparison of batch learning on category-level during training and testing on the new data instances for different approaches. The results are averaged across five learning trials.

Approach	**Accuracy (%) (Training)**	**Accuracy (%) (Testing)**
GDM with TC	88.12%	81.67%
GDM without TC (test)	86.83%	79.82%
GDM without TC	83.68%	80.07%

TC). Based on the results it is clear that learning the temporal relation of the input plays important role in increasing the performance of the model at both instance and category levels.

From the results, we can see that the model not only performs better in training data but it also shows good generalization results on the test dataset. From the figure 4.5(a), it can be observed that the growth of the neurons is stabilized after 20 epochs for both episodic and semantic memory. This indicates that the neurons are habituated for the given input data. It can be observed that the number of neurons in episodic memory is higher than semantic memory. This is expected since the G-SM network grows only when the predicted BMU label is misclassified with the input data label, whereas G-EM network growth is unregulated as it grows when the new input data distribution is observed based on the activation threshold a_T^{EM} and habituation threshold h_T. As the number of neurons increased and habituated to the input data, we can see that the average quantization error in figure 4.5(c) is significantly reduced and the accuracy over the epochs also getting stabilized (figure 4.5(b)). From the result of individual object category accuracy in figure 4.5(d) on the test scenes, we can see that most of the objects are classified with good accuracy values. Except for the few objects like pencil, fork, and USB where the prediction performance is comparatively less than other object categories. This may be due to the existence of identical object categories like spoon, pen, staple, and eraser. Even though we observed performance drop for those objects, it is to be noted that the GDM model can able to identify and distinguish the objects with similar shapes and sparse representations.

4.3.2 Incremental Learning

In incremental learning, the training samples consist of objects from 25 categories that are progressively available over time in the form of mini-batches. Where each mini-batch contains samples from all 12 training scenes based on the different object instances and categories. Here, the number of epochs will be equal to the number of object categories, where the object that belongs to the particular category is used once to train the model and it is not shown again while learning new categories. The performance of the network in alleviating the catastrophic forgetting is evaluated by using the recursive reactivate neural activation trajectories (RNAT's) and intrinsic memory replay. Then the results are compared between models trained with and without using memory replay. Similar to batch learning, the data from scenes s6, s8, and s14 are used for testing, while the point cloud data samples from the remaining scenes are used for model training. The hyperparameter values for certain constants are used similar to the batch learning scenario expect for the activation thresholds of G-EM (a_T^{EM}), and G-SM (a_T^{SM}), the learning rate of BMU (ξ_b) and neuron (ξ_n), and the global context (β).

The hyperparameter settings used for the incremental learning are shown in the table 4.3. Based

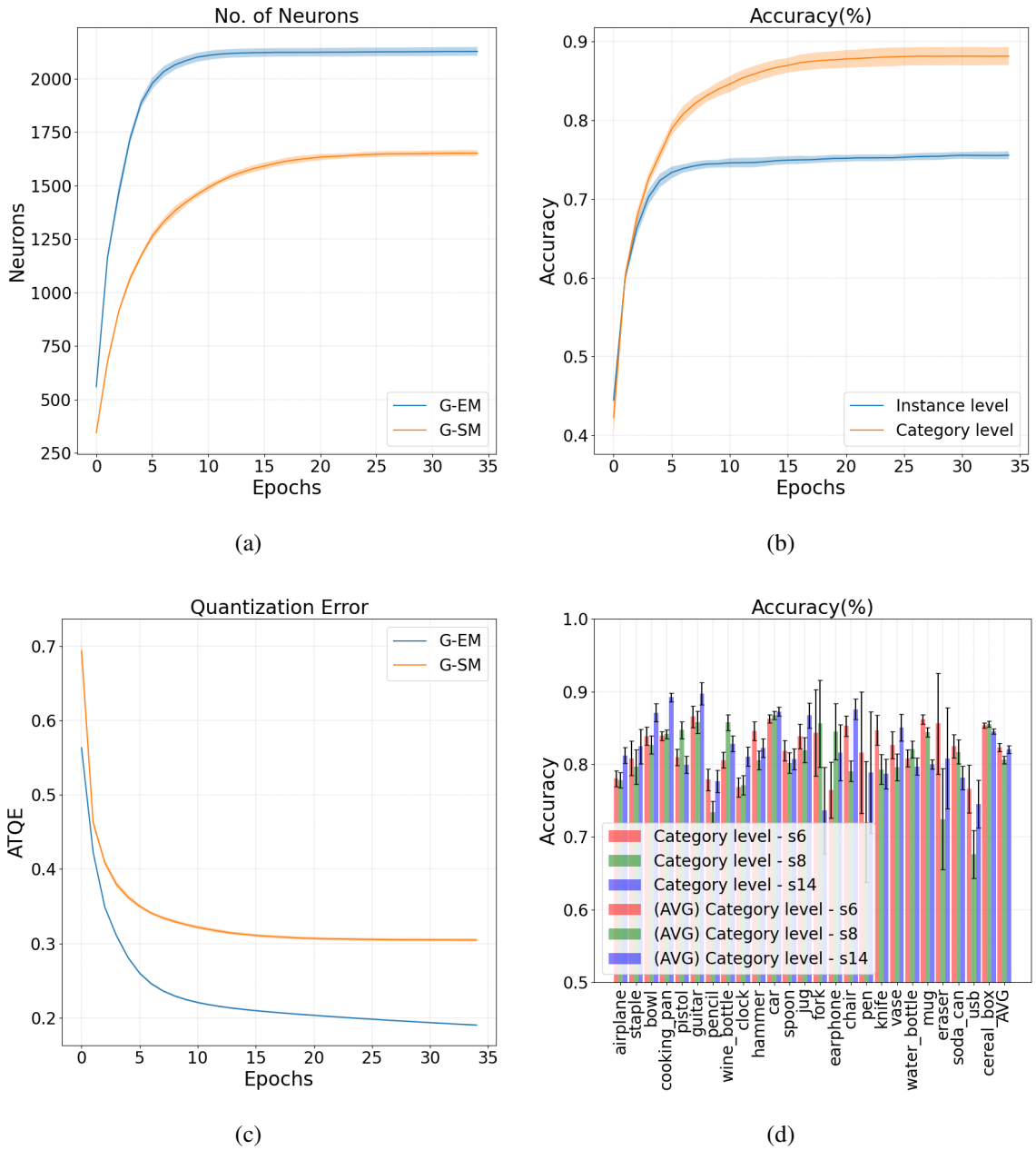


Figure 4.5: Batch learning results: (a) Number of neurons in episodic and semantic memory, (b) Accuracy of G-EM and G-SM on instance and category level, (c) Quantization error, and (d) Accuracy on the test dataset. Results are averaged for five learning trials, the shaded area shows the standard deviation.

on the different experimental trials, the hyperparameter values shown in table 4.3 give the best results to the generated synthetic dataset for incremental learning. The effects of change in hyperparameter values can be noticed in the effect of neuron growth by comparing the batch and incremental learning results, figure 4.5(a) and figure 4.6(a). At the end of every learning epoch, the accuracy is estimated for all the object categories encountered so far in the incremental learning task i.e., the accuracy is estimated including the data samples from already learned categories while estimating the new task. For example, if the network is training on the third object category (i.e. at third epoch) its accuracy is calculated including data samples from the category 1 and 2. In this way, the performance of the model can be estimated to check whether the model learns new object categories without forgetting

the previously learned ones. The training procedure for incremental learning is presented in algorithm 6.

Algorithm 6: Training procedure for incremental learning

1. Initialize the hyperparameters;
2. Get $data$, e_{labels}, and s_{labels} for the incremental training using the data sampling algorithm 3;
3. Initialize the G-EM and G-SM networks;
4. For n number of object categories, train G-EM network with $data$, e_{labels}, and s_{labels} using the Gamma-GWR algorithm 1;
5. Get episodic weights and episodic labels from G-EM training;
6. Train G-SM network with episodic weights and episodic labels using Gamma-GWR algorithm 1;
7. **if** *Memory replay enabled* **then**
   - 7.1 Replay G-EM neurons using equation 14 and 15 with replay window size, $\lambda = K^{EM} + K^{SM} + 1$;
   - 7.2 Get replay weights and replay labels;

   end
8. **if** *Memory replay enabled and number of categories > 1* **then**
   - 8.1 Train G-EM network with the replay weights and replay labels using Gamma-GWR algorithm 1;
   - 8.2 Train G-SM network with the replay weights and replay labels using Gamma-GWR algorithm 1;

   end
9. Compute the test accuracy for number of categories encountered so far;
10. Repeat step 4-9 until all the objects categories are trained;
11. Compute the final test accuracy of G-EM and G-SM networks;

Results and Interpretations: Table 4.4 shows the result of incremental learning with and without memory replay. The results shown are trained over 25 epochs and average across five learning trials. From table 4.4, GDM with intrinsic memory replay (MR) gives the best overall average accuracy.

Table 4.3: Hyperparameter settings for G-EM and G-SM networks in incremental learning.

Hyperparameter	**Values**
Insertion thresholds	$a_T^{EM} = 0.5, a_T^{SM} = 0.7$
Global context	$\beta = 0.4$
Learning rates	$\varepsilon_b = 0.5, \varepsilon_n = 0.005, \varepsilon_c = 0.001$
Habituation threshold	$h_T = 0.1$
Habituation function	$\tau_b = 0.3, \tau_i = 0.1, \kappa = 1.05$
Edges removal threshold	$N_T = 0.2$
Labelling	$\delta^+ = 1, \delta^- = 0.1$
Context descriptors	$\alpha_1 = 0.63, \alpha_2 = 0.234, \alpha_3 = 0.086$

The model trained with GDM using memory replay gives overall 40.42% accuracy at the instance-level, 62.25% category accuracy on the training validation, and 59.33% overall accuracy when tested on the test samples. When comparing the results of the model trained on GDM with and without memory replay, we a observed 6.03% reduction in instance accuracy while the category level accuracy increased by 4.8%. This increase in category accuracy is also observed when both the models tested on the unseen test samples, we observed that the model with MR gives 2.16% accuracy improvement over the model without MR.

Table 4.4: Classification performance of incremental learning at category-level during training and testing for different approach. Results are averaged across five learning trials.

Approach	**Accuracy (%) (Training)**	**Accuracy (%) (Testing)**
GDM with replay	62.25%	59.33%
GDM without replay	57.45%	57.17%

Figure 4.6 shows the number of neurons, accuracy, and the quantization error of the incremental learning scenario over 25 epochs average across five learning trials. Unlike the growth of neurons in batch learning (figure 4.5(a)) at the initial epoch where the neurogenesis is particularly strong. In incremental learning, we observed neuron growth (figure 4.6(a)) progressively over time in both the networks when the new novel objects are exposed to the model. We also observed a monotonically increase and decrease in quantization error (figure 4.6(c)) when the new object category is introduced to the model in both the networks. As expected the overall accuracy in both instance and category decreases as the new object categories are introduced. We observed the sudden drop in instance and category accuracy when the staple object (2nd category) is introduced after the model is trained with the airplane object. This may be due to that the airplane object has high-level features compared to the staple object, but after learning the staple object we observed the increase in category accuracy. We also noted similar behavior during the learning process of spoon, pen, and eraser objects. This clearly shows that model is sensitive to different features levels (i.e., object with rich features and objects with similar features (pen and pencil or spoon and fork)). Also, we observed that the order of object categories has higher sensitivity towards the model performance.

Comparison: Figure 4.7, shows the comparison plot between the model with and without memory replay in episodic and semantic memory. When comparing the neuron growth between the models, (figure 4.7 (a) and (b)) the memory replay influences the neuron growth in both episodic and semantic memory. In contrast to the batch learning and the model without memory replay, the number of neurons in semantic memory is higher than episodic memory in the model with memory replay for our selected dataset. This change in behavior with memory replay may be due to the addition of external noise in the dataset. Sometimes additional external noise to the point cloud may change the structure of the object, which results in a completely different and new feature representation. This results in high neuron activity in the GDM networks and during the memory replay phase due to high-level neuron activity new nodes are added. Since there is no periodic replay in the model without MR less neuron growth is observed. When comparing the accuracy results (figure 4.7 (c), and (d)), the instance accuracy of both the models remains more or less the same, whereas the memory replay model shows a significant improvement in category accuracy for all the learned object categories. Based on our

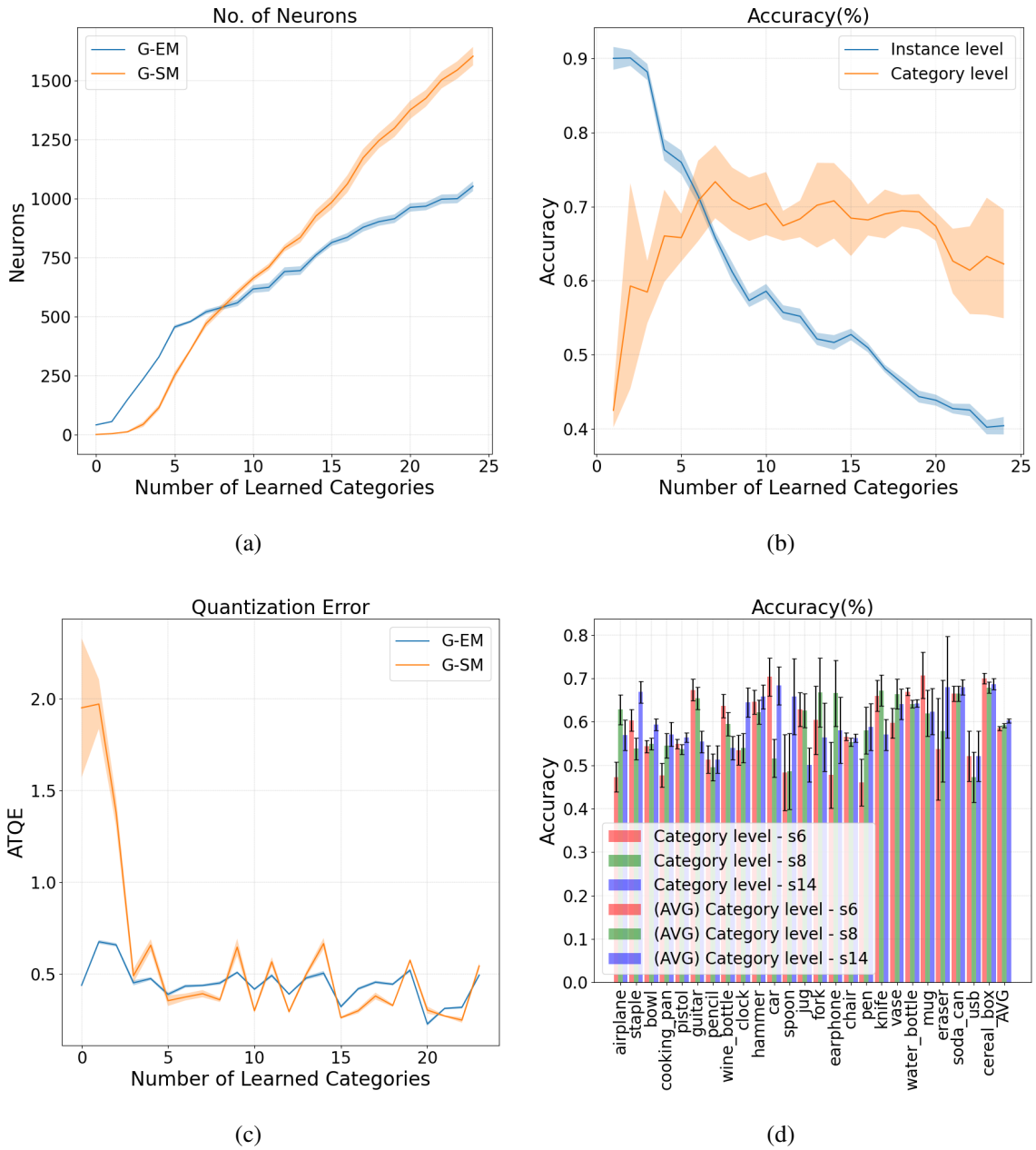


Figure 4.6: Incremental learning results: (a) Number of neurons in episodic and semantic memory, (b) Accuracy of G-EM and G-SM on instance and category level, (c) Quantization error, and (d) Accuracy on the test dataset. Results are averaged for five learning trials, the shaded area shows the standard deviation.

experimental results, the intrinsic memory replay helps to improve the model performance in incremental learning scenarios by the periodic replay of temporal trajectories (RNATs) learned from G-EM memory. When interpreting the category level accuracy of the GDM model with memory replay (see fig.4.6(b)), we can observe that the model retrains the previously gained knowledge while learning the new categories. This shows that a model with intrinsic memory replay mitigates the problem of catastrophic forgetting.

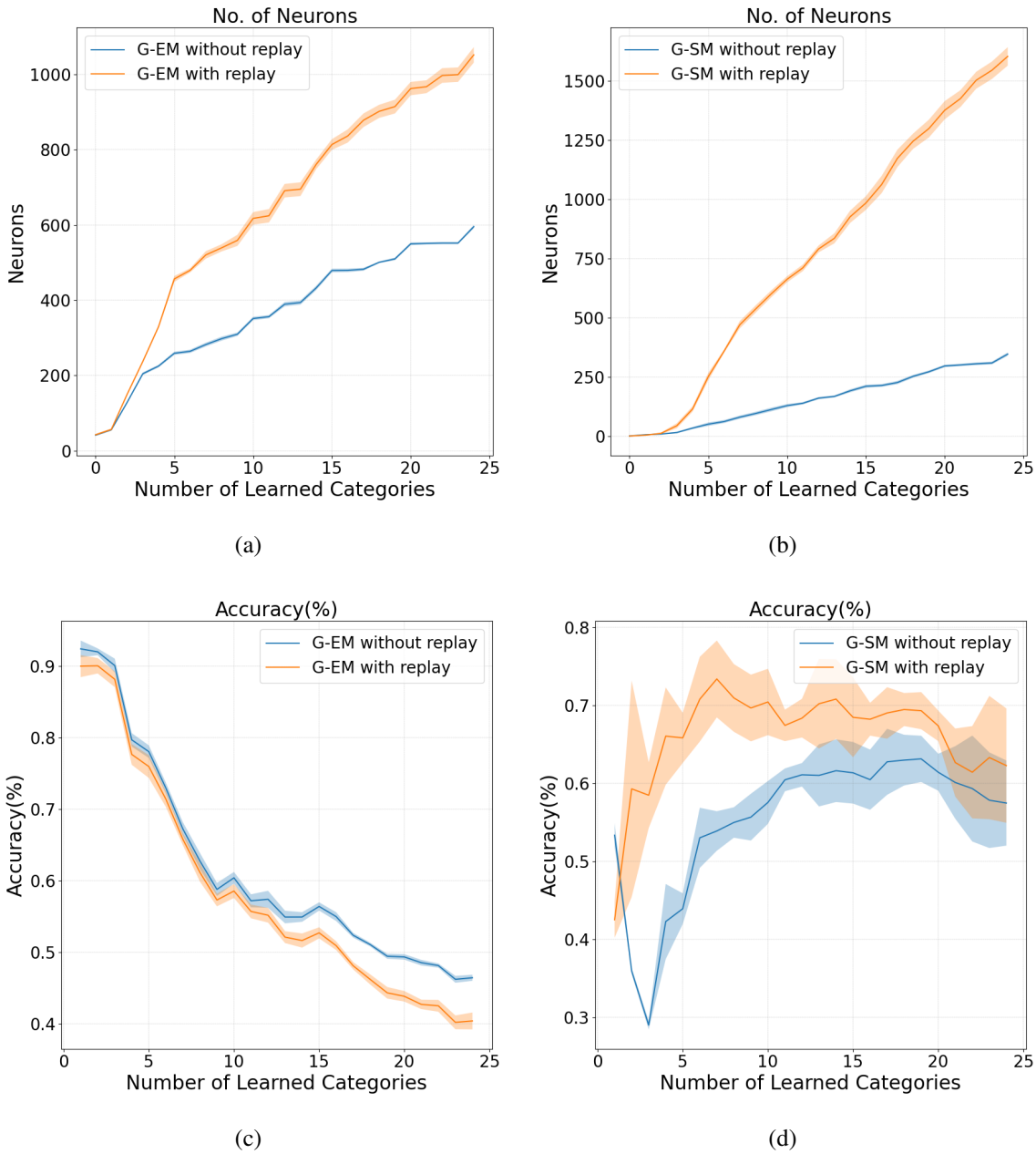


Figure 4.7: Comparison results of incremental learning with and without memory replay: (a) Number of neurons in episodic memory, (b) Number of neurons in semantic memory, (c) Episodic memory accuracy, and (d) Semantic memory accuracy. The results are averaged across five learning trials.

4.3.3 Continuous Object Recognition

We evaluate the incremental learning model with the three continuous object recognition scenarios proposed by the CORe50 benchmark dataset. The learning tasks include new instances (NI), new classes (NC), and new instances and classes (NIC). In NI settings, the model is initially trained with data of known categories then the new instances that belong to the same object class from different acquisition scenes are available over time. Hence, the model must be dealt with in learning the new input sensory experience and predict them based on different object categories. For the NI approach, the model is trained with the first training scene and further incrementally trained with the remaining

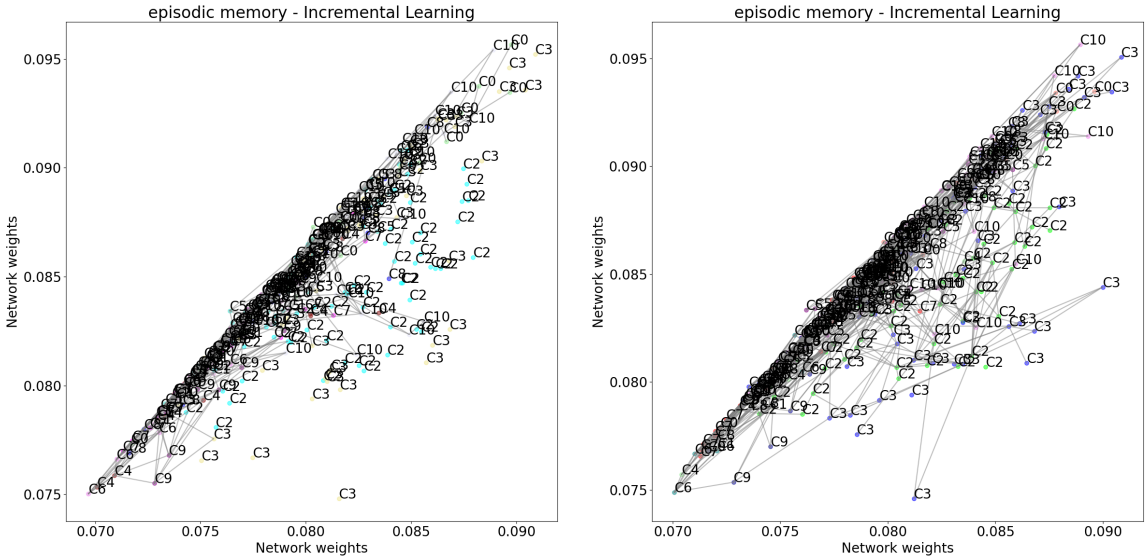
Table 4.5: Accuracy of continuous object recognition on the incremental learning scenario.

Approach	**Accuracy (%)**
NI - GDM with replay	54.95%
NI - GDM without replay	49.13%
NC - GDM with replay	45.86%
NC - GDM without replay	62.64%
NIC - GDM with replay	56.98%
NIC - GDM without replay	61.05%

11 scenes. In NC, new classes belong to the different object categories are progressively available over time. Hence, the model must be dealt with learning new object categories while retaining the knowledge about previously learned categories. For the NC scenario, the model is trained with four training mini-batches. In the first batch, 10 object categories from all the training scenes are included and the remaining three batches include 5 object categories each. In the NIC setting, samples belong to new instances and categories became available over time, requiring the model to learn new ones while retaining the previously learned ones. The NIC training consists of 20 mini-batches, created from 11 scenes and 25 object categories. The first batch includes 10 classes to maximize the categorical representation and the remaining batches contain objects from 5 classes with only one training sequence per class is included. The classification accuracy of continuous batch learning on the test samples is shown in table 4.5. The test set from the test dataset (s6, s8, and s14) contains all the samples from all the classes to keep consistency across all three different learning scenarios. Similar to the experimental observation in incremental learning (shown in table 4.4), in the NI scenario, the model with memory replay gives better accuracy than the model without memory replay. To contrast in the NC, and NIC scenarios the model without memory replay gives better results showing the need for performance improvement. We also tested the continuous object recognition on the unknown categories (i.e. apart from the 25 object categories used during model training) during our simulated robot testing. Even though the model prediction accuracy with continuous object recognition is less, during our robot simulation testing the incremental learning model performed well at learning the new unknown object categories and classifies them perfectly.

4.3.4 Effect of Controlled Connections Removal

During the initial experimentations with GDM learning [21] on batch and incremental learning scenarios, we observed that at the end of learning, neurons with good habituation value (i.e., the neurons which represents good knowledge about a particular object category) is being removed due to its maximum age criteria and no neighboring connections. Since the neurons are removed despite their knowledge level about the input data we observed an accuracy drop in episodic and semantic memory predictions. We observed a huge accuracy drop in the episodic memory than in the semantic memory. This is because the semantic memory neuron growth is regulated (i.e, G-SM adds new neurons only when the BMU label prediction is misclassified with the ground truth data) whereas the episodic memory is unregulated (i.e., the neuron grows whenever new input activity is observed), which results in more number of neurons are getting removed in episodic memory at the end of learning thus results in poor prediction accuracy. Figure 4.8(a) shows the network representation of episodic mem-



(a) (b)

Figure 4.8: The network plot of episodic memory with and without controlled connections (edges) removal. (a) without controlled connections removal, and (b) with controlled connections removal. The plot shown above contains the network representation for 11 categories ($c0 - c10$). The scatter plot shown are the 2D representation of network weights, where the dimension of network weights (i.e., 256 dimension) are reduced using principal component analysis (PCA).

ory with neighboring connections removed at the end of 11th epoch. From the figure 4.8(a) we can see that most neurons representing the class $c2$, and $c3$ are left unconnected due to their maximum age despite its knowledge level in the network. To solve this issue we proposed the controlled edges (connections) removal technique based on the neurons knowledge level (by the habituation value h_i of neuron i), the detailed explanation of this technique is explained in section 3.2.6. Figure 4.8(b) shows the result of controlled edges removal at the end of 11th epoch training (similar to 4.8(a)). From the result of controlled connections removal (see fig.4.8(b)) we can observe that by the influence of N_T threshold, most of unconnected $c2$, and $c3$ neurons from figure 4.8(a) are retained. By comparing the figures we can see that most of the neurons with a good amount of knowledge are removed by the baseline GDM algorithm [21] but with our proposed connection removal strategy, those neurons are retained and adapted well to the input data distribution over time (t) by finding their neighbors and moving close to their associated neighbors. Here, to recall if the habituation value of neuron i (h_i) is 1 represents that neurons are newly created and it has no knowledge about the input data, whereas h_i value is close to 0 means that neuron i has high-level or good knowledge representation about the input data.

We experimented with different choice of N_T (edges removal threshold) parameters, we tested N_T values of 0.1, 0.2, 0.3, 0.4, 0.5, and 0.6. The effect of GDM performance on episodic and semantic memory based on different choices of N_T threshold is shown in figure 4.9. This figure also contains the plot of the GDM model result with no controlled connections removal (no N_T), for comparison. The dotted lines in figure 4.9 indicate the result of N_T where as the dashed line indicates the result of no N_T, the dotted line with triangle indicates the best performing result ($N_T = 0.2$). Form figure 4.9 we can see that N_T value with 0.2 gives the overall best performing results. When comparing the effect of N_T in the episodic memory (figure 4.9 (a) and (b)), the no NT model gives the low instance-level prediction accuracy (pink line) and fewer neurons growth, whereas the N_T helps to improves the prediction accuracy overall for all the different values. In the episodic memory the N_T values, 0.2

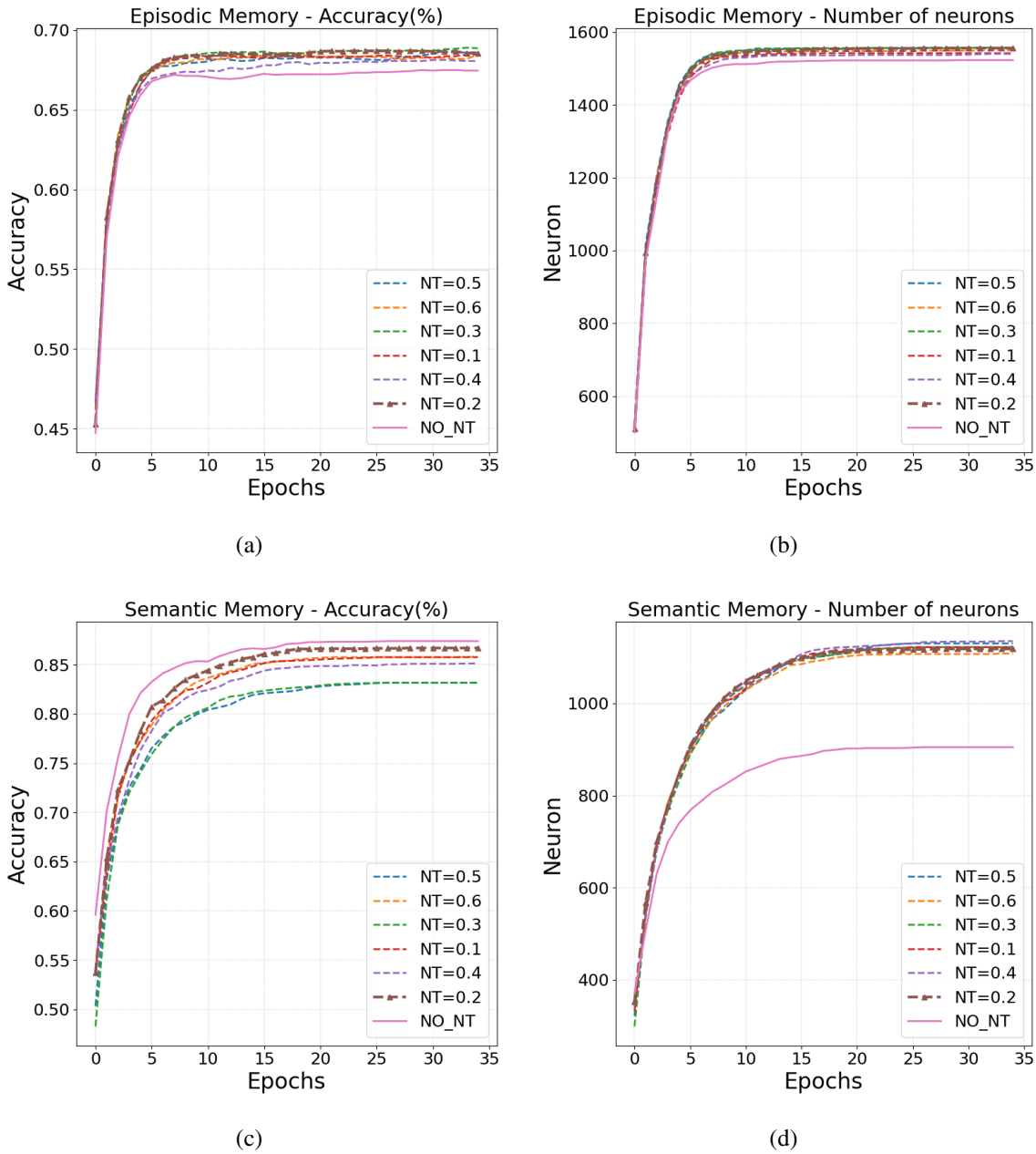


Figure 4.9: The effect of controlled edges (connections) removal on different N_T threshold values on: (a) episodic memory accuracy, (b) neurons growth in episodic memory, (c) semantic memory accuracy, and (d) neurons growth in semantic memory over different learning epochs. The dashed plot with triangles ($N_T = 0.2$) shows the best overall performance.

and 0.3 give the best instance-level prediction accuracy's when compared with other values. When interpreting the results of N_T influence in the semantic memory (figure 4.9 (c) and (d)), in contrast to episodic memory the GDM model with no N_T gives the best category-level prediction accuracy and less neuron representation when compared with the models trained with N_T. The increase in accuracy may be due to the low neuron growth of the GDM model without N_T, which reduces the complexity during the accuracy estimation. Overall, we are interested in the N_T value which gives considerable neurons growth and achieves good prediction performance. The GDM model with N_T value of 0.2 satisfies the above needs when compared with other values. Though the GDM model with N_T value 0.3 gives the higher number of neurons representation, its prediction performance is low (figure 4.9

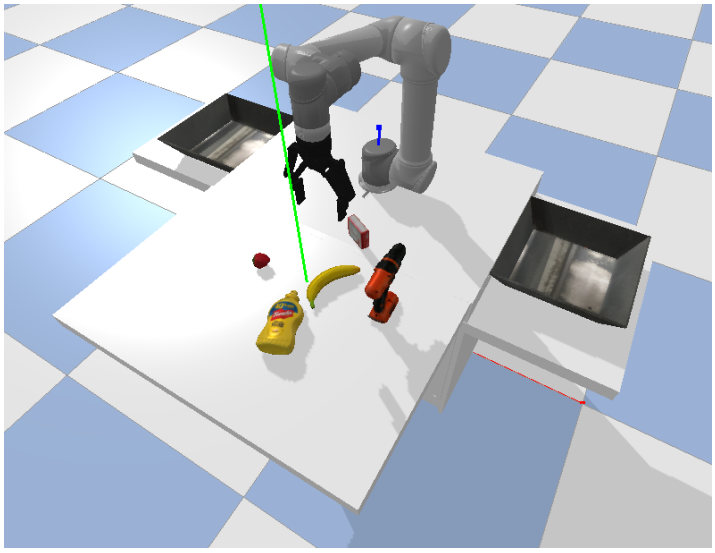


Figure 4.10: Experimental setup for the simulate robot experiments, consists of URe5 robot arm and RGB-D camera from the pybullet environment. This simulated environment is the adapted and modified work of [4]. The objects used in this experiment are imported from YCB dataset [5]. The green line at the center indicates the camera line of sight.

(c), green line). Only the model with N_T values of 0.2 gives the accuracy values close to the model with no N_T. When comparing the overall results of figure 4.9, even though the model with no N_T excels at performance on semantic memory, the model with N_T value of 0.2 best performance at both episodic and semantic memories. Hence, we used the N_T value of 0.2 for training the GDM model on all the batch and incremental learning scenarios. To note that the results shown in figure 4.9 are the results of the GDM model trained on batch learning with no temporal context, we choose this model because it trains faster compared to other models.

4.4 Evaluation on the Simulated Robot Environment

We evaluated the performance of the real-time object classification and prediction in the simulated robot environment. The task includes pick and place scenarios in the context of clear the table, and pack scenario. In both scenarios the robot needs to predict the object categories before performing the intended tasks.

4.4.1 Environmental setup

The simulated robot environment is developed in the pybullet physics engine. The simulated environment used in this study is the adapted and modified work of *J. O. Vrielink and H. Kasaei* [4]. The modified simulated environment setup consists of a URe5 robot arm with two-fingered grippers placed on top of the table, and two baskets (basket1 - to the right side of the robot arm, and the basket2 - to the left side of the robot arm). During experiments based on the manipulation scenarios, the objects are placed in the respective baskets. Figure 4.10 shows the environmental setup for the simulated robot experiments.

4.4.2 Experiments and Results

The simulated robot experiment consists of the end-end implementation of the proposed system architecture (see fig.3.1). The step-by-step process of the end-end system on the simulated robot exper-

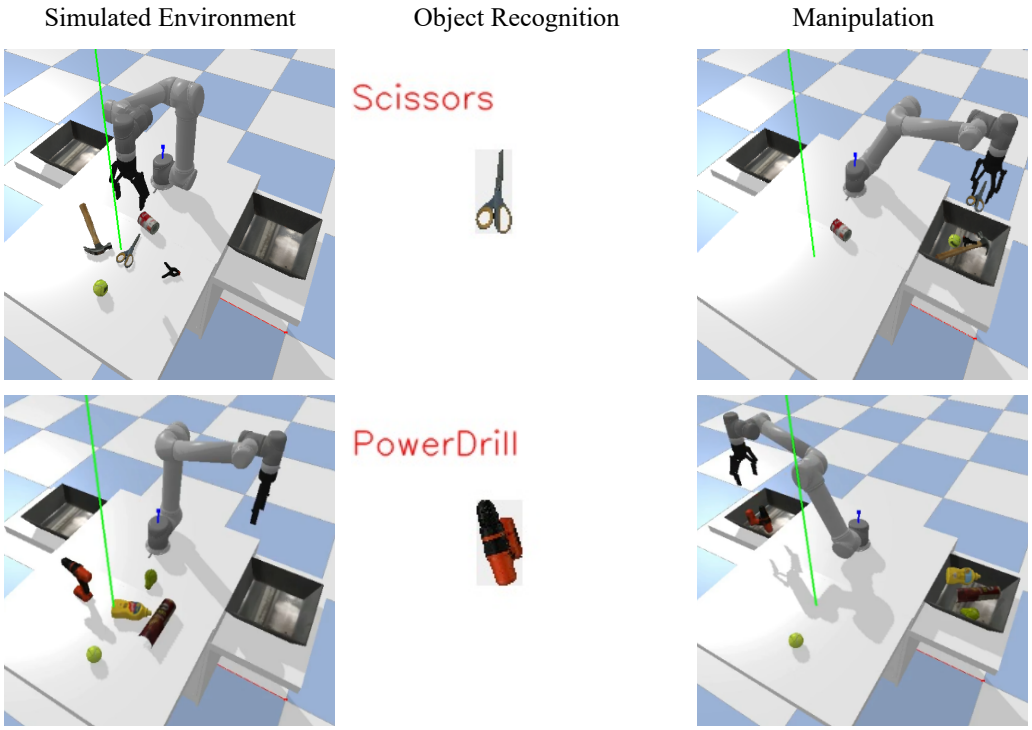


Figure 4.11: Simulated robot experiments: (i) simulated environment with objects (first column), (ii) object recognition result (second column), and (iii) manipulation of the object by picking and placing it to the basket (third column).

iment is as follows, the RGB and depth image view of the object (at current position and orientation) received from the camera is converted to an RGB-D image. Then it is passed to the autoencoder network to obtain the object representation and the grasp configuration. If more object samples are present, the RGB-D view with the maximum entropy is used to generate the grasp configuration. The generated object grasp configuration is then used to estimate the grasp points for object grasping. Then the predicted object representations are passed to the GDM network to perform the instance-level and object-level classification. If the new object category is exposed, based on the simulated teacher input (figure 3.1) the robot learns and recognizes the exposed new object category or instance. When the new object categories are encountered during training, the label belongs to that object category is updated to the existing known objects list. To learn the new object categories we sampled 30 RGB-D data points by changing the position and orientation of the 3D object in the environment.

To detect object boundaries, we used contour-based bounding box approximation, in all the simulated experiment scenarios. We re-scaled each object detected in contour boundaries to 224×224 to match the input dimensions of the autoencoder model. We used the incremental learning GDM model with memory replay for the object prediction and classification, we also used the same hyperparameter values that were used in the incremental learning scenarios (Table 4.3).

In the pick and place experiments, the robot needs to recognize and grasp the object in the environment to place it in the basket. The pick and place experiments are set up in a way that the robot needs to recognize the object first before grasping, and manipulation. In the pack scenario, two baskets (named basket 1, and basket 2) are placed in the environment, where the object with the selected category (e.g., Scissors) needs to be packed in basket 1 and the object that belongs to the remaining category needs to placed in basket 2. If none of the object categories are selected all the objects need to be placed in basket 2. The performance of the experiments is assessed by calculating the success rate, i.e. $\frac{Number of success}{Total number of attempts}$. Note that the experiments will be counted as a success only when the

objects are placed inside the basket.

We imported 15 simulated objects from the YCB dataset [5] for simulated robot testing. For the pick and place task, we tested each simulated object 50 times. For each learning experiment, we randomly placed the objects on the table.

As a result, for the pick and place task, we achieved an 80.27% success rate (i.e., 602 success out of 750 attempts). For the pack scenario, five objects are randomly placed in the table in which the selected object category (based on the user input) needs to be placed in basket 1 and the rest in basket 2. Here, the success will be counted only when the objects are placed in their respective baskets. We conducted the pack experiment for 10 times, where each experiment have randomly shuffled object categories. For the pack scenario, we achieved the grasp success rate of 68% (i.e., 34 success out of 50 attempts). For most experiments in the pack scenario, even though the model predicts the correct object labels due to an inaccurate bounding box the robot couldn't place the object to the target location. In common for both pick and place, and pack scenarios, some attempts are failed in conditions such as objects attached to the gripper get slipped while reaching the basket, misclassification of the target objects, the collision between the objects to the robot gripper, and incorrect pose estimations. Figure 4.11 shows the results for simulated robot experiments, scissors object in top-row, and power drill object in bottom-row. The results shown for the power drill object is an example of pack scenario, where the robot needs to pack the selected object (i.e., power drill) to basket 1 (basket to right-side of the robot arm), and the remaining objects to basket 2 (basket to left-side of the robot arm). From the results, we can see that the proposed system architecture shows good real-time performances on the simulated robot environment with object recognition and grasping. The results also show that the GDM model learns and adapts when a new object instance or category is exposed. It can be also inferred that model is performing well at predicting the antipodal grasp points for the unseen 3D objects.

5 Discussion and Conclusion

In this chapter, the conclusion of this thesis work is drawn. First, we discuss about the shortcomings and challenges that were encountered during this work. Then, the research questions proposed in chapter 1 are answered based on our experimental findings. Finally, we summarize the our work by the brief conclusion, and remarks for the possible future directions to improve the work.

5.1 Discussion

Even though we observed that model preforms good at object recognition learning and grasping simultaneously, we observe certain shortcomings in the proposed model. The following subsections explains them in detail.

5.1.1 Features and Similarity measures

In the proposed system we used an autoencoder that extracts the features and grasp map based on the point cloud based RGB-D images. In the experiments of grasp synthesis learning and the GDM model on batch learning, we observe that the model performs better with good prediction accuracies. But in the experiments of GDM learning in incremental learning strategy, we observed the drop in model performance. This is due to the choice of non-identical objects with similar features that exist in the scenes. For example spoon and fork, pen and pencil, eraser, USB stick, and staple objects. These objects belong to different object categories but are mostly similar in geometrical shapes, which increases the complexity of the model to predict and differentiate the features among them.

From the results of incremental learning in figure 4.6 (b) and (d), and in figure 4.7 (d), we see that while learning the above-mentioned objects the model accuracy is dropping and starts increasing while learning the other object categories. This behavior is not observed in a batch learning scenario, because in batch learning the model is trained with entire object categories, hence the result computed over the epochs are estimated for all the object categories. But in an incremental learning scenario, this is not the case, here the object that belongs to a particular category is trained and evaluated on the number of encountered categories (explain in section 4.3.2). Even though the influence of object category is not encountered in validation accuracy of the batch learning (see fig.4.5 (b)), we observed its reflection while testing on test dataset for individual object category accuracy (see fig.4.5 (d)). From the batch learning result on the test dataset in figure 4.5 (d), the accuracies for pen, pencil, USB, fork, and eraser objects were comparatively less than the other objects. This clearly shows that the non-identical objects with similar shapes play a huge role in the model performance. Even though our model predicts them with low accuracies we can see that our model can identify and distinguish those fine-grained input data distributions.

The another reason for the drop in prediction performance of GDM models on point cloud based data samples may be due to estimation of BMU's using distance-based similarity measures. In our 3D object selection for the dataset generation and real-world, as we discussed in the above passage the objects belong to different object categories with similar features exists (e.g., spoon and fork). On such occasions, while estimating the distance between those objects (spoon and fork) they may seem close to each other while comparing it with other objects (e.g., water bottle). We observed this behavior in our GDM model learning. While testing the trained incremental GDM model on the test dataset we observe that samples belong to the pen object are predicted as pencil due to its similar structure and most similar feature representations with pencil objects. This indicates the need for object-based similarity measures to distinguish these fine-grained representations. Even though the

model performs is less at the objects with similar shapes, the overall accuracy of the GDM models is retained and it good at predicting the objects from different categories.

5.1.2 Training time

The training time of the GDM learning model differs based on the different learning scenarios and approaches. In our experimentation, we observe that the GDM model with and without TC on batch learning and the GDM model without memory replay on incremental learning trains faster compared to the incrementally trained GDM model with memory replay. For the GDM model without TC ($K = 0$) on batch learning, the training complete within an hour across different learning trials. For the GDM model with TC on batch learning, it took approximately 2 hours 30 minutes to train the model across different learning trials. This increase in training time is due to the additional learning of temporal context (TC) during the learning process in the model with TC. For the incremental learning scenarios, the GDM model without memory replay took approximately 1 hour 30 minute for different learning trails. Even though the model with TC on batch learning and model without memory replay on incremental learning learns TC during learning, the representation of input object categories from different scenes influenced the training time.

In the GDM model with incremental learning, the object belongs to a particular category shown once at a time, unlike the entire object category representation in batch learning. This results in less training time on the semantic memory over epochs and thus resulting in a faster training time on the incrementally trained GDM models. But for the GDM model with memory replay on incremental learning, single learning trials took approximately 12 hours 30 minutes to train the model for 25 object categories. The intrinsic memory replay of the episodic memory at each learning epochs increases the training time. Since the neurons in the episodic memory grow when the new object category is exposed over the epochs (i.e., number of object categories), which results in the exponential increase of the replay time during intrinsic memory replay. The replay time depends on the number of neurons in the episodic memory. To reduce the training time of the GDM model with memory replay, instead of replaying all the neurons in episodic memory at every learning epoch we need a technique that replays the selective neurons with RNATs in episodic memory during the intrinsic memory replay, and at the same time, it needs to be good at retaining the classification performance.

5.1.3 Object Detection

We used contour-based object detection techniques while testing the real-time performance of the proposed system in the simulated robot experiments. The contour area threshold in separating the object boundaries plays a huge role in both object recognition and object grasping. When the objects are placed close to each other in the environment, the object detection pipeline results in detecting the two or more objects in the single boundary. Which results in choosing the contour area threshold for different objects. As the result of incorrect bounding box detection, the GDM model results in incorrect classification and the autoencoder results in bad grasp map prediction. In future work, the contour-based object bounding box detection needs to be replaced with 3D objects based bounding box detection using point clouds. We already have an implementation of point cloud based bounding box detection for 3D objects in our dataset generation pipeline (see fig.3.2) in the ROS environment, which can be utilized in future work. Since our simulated robot environment is in pybullet we are not able to use that functionality.

5.2 Answers to Research Questions

Does the sequential point cloud representation of a 3D objects helps to learn the object recognition and grasp affordance in continual learning fashion?

From the experimental results presented in chapter 4, it can be seen that the proposed hybrid learning architecture consists of GDM networks and generative autoencoder learns and predicts the 3D object recognition and grasping for the proposed synthetic sequential point cloud dataset (section 3.1). From the results of incremental learning experiments and continuous object scenarios, the proposed model can continually learn the new object instances while retaining the previously gained knowledge. From the results we can also see that, the system not only performs better at classifying the objects with different shapes, it can also be able to recognize and classifies the objects with similar shapes (e.g., pen and pencil) from the sequential point cloud samples. Even though we observed a drop prediction performance for similar objects, but still the GDM manages to learn the fine-grained representation and performs the classification. To solve this prediction accuracy issue as we discussed, using object-based similarity measures instead of distance-based measures will be a viable option.

Is the point cloud representation of the input data helps the GDM algorithm to learn, and classify the object categories? How does the modified GDM learning helps to improve the performance?

From the results of GDM learning in section 4.3 we can see that with the object representation predicted by the autoencoder model for the sequential point cloud samples at a time (t), the GDM model can learn and predict the object class at an instance and category level in batch and incremental learning scenarios. From all the experimental results, the GDM model with temporal context (TC) on batch learning shows the best performance results (88.12% during training, and 81.67% during testing). In incremental learning scenarios, the GDM model with memory replay gives the best performing results (62.25% during training, and 59.33% during testing). In addition to that, the introduced controlled connection removal in the Gamma-GWR learning in both episodic and semantic memory shows the improved performance results, which indicates that the proposed modifications to GDM learning have a significant role in improvising the prediction performance of the GDM models.

How does the proposed system continually learn the new object categories, and perform simultaneous grasping in real-world scenario?

The experimental results of the real-time evaluation of object recognition and grasping in section 4.4 for pick and place, and pack scenarios show us that the proposed system can continually learn the object categories and at the same time it can perform object grasping. From the results, we obtained an 80.27% success rate for pick and place scenarios, and a 68% success rate for pack scenarios. Since the objects used in the simulated robot testing is from different 3D object dataset, we can see that the GDM model can continually learn the new object categories and the autoencoder model can generate the grasp points for new 3D objects. These results also show that the simulated teacher in the proposed system can collect the samples from the environment efficiently to train the new object categories. In the future, this simulated teacher can be replaced by the robot actions (e.g., motor actions), so that the robot can autonomously capture the different object views from its vision of sight to learn the newly encountered objects in the environment autonomously.

5.3 Conclusion

In this research work, we presented a machine learning model using a growing dual-memory network (GDM) and generative autoencoder to learn the 3D object recognition and grasping using the synthetic sequential point cloud dataset on lifelong learning fashion. The experimental results by GDM on the batch learning and the incremental learning implies that the GDM model can learn the object recognition using the point cloud-based RGB-D data at the instance and category level. GDM in the incremental learning setting demonstrates that our model can recognize and classifies the non-stationary data that is progressively available over time. The problem of catastrophic forgetting is addressed by the intrinsic memory reply using RNATs, based on the temporal knowledge learned during the training. The results from the autoencoder also demonstrate that the proposed model can predict accurate grasp maps and compact object representations. The storage complexity in the continual learning problem is also addressed in this research work. At end of learning, the batch learning model (GDM with TC) utilizes storage space of 132.6MB for episodic memory and 84.2MB for semantic memory. In the incremental learning (GDM with memory model) the episodic memory utilizes 38.8MB storage space and the semantic memory utilizes 75.4MB storage space. The GDM models do not need explicit storage of old data while learning the new ones, by which the system can learn from the direct vision without the need for external dataset storage. This type of system can be implemented in real-world robotic systems where there is no or less external supervision is required, with the capability of learning the new objects continuously and performing simultaneous object grasping, this system can be used for different manipulation tasks in different domains.

5.4 Future Work

The proposed system for continual learning of 3D object recognition and grasp synthesis prediction using the generative autoencoder and GDM learning in the thesis proves that the system can continually learn the new object categories in real-time and simultaneously it can predict the grasp maps for the given object and performs the intended manipulation tasks. Even though the proposed model address continual learning with limited storage complexity, still a lot of improvements and enhancements need to address to make it work efficiently. Especially lot of attention needs to be focused on improving the prediction performance of the incremental learning GDM model. The choice of hyper-parameters for incremental learning needs to be further investigated and fine-tuned to improve the performance of the GDM learning. As discussed above, this reduction in performance is due to the addition of objects with similar shapes and adding noise in the learning process. This problem can be addressed in future work in two ways. *(I)*. Instead of using the distance-based similarity measures to predict the best matching unit (BMU) in the GDM networks object-based similarity measures can be employed, if this option seems viable the neuron activity function ($a(t)$) needs to be readdressed to make the object-based similarity measures to work. *(II)*. The other possible future direction is to use point cloud based feature extractors. Even though the GR-ConvNet model predicts the grasp rectangle based on point cloud data in our approach it uses RGB-D based images to predict the object representation and grasp maps. It gives a better grasp map prediction and object representations, but this can be further improved by using the feature extraction on the point cloud (3D) level instead of on the image level (2D). We can adapt encoder-decoder based feature extractors and grasp affordance prediction models like Res-U-net [54] to increase the likelihood of point cloud neighbor representation in GDM networks.

In this thesis work, all the point cloud samples are synthesized from the simulated 3D object, in further we would like to generate and test the learning using a synthetic sequential dataset based

on the real-world object which has enriched feature representations than the simulated objects. One problem that needs to be addressed is the learning time of the GDM model with intrinsic memory replay. Since during replay all the neurons inside the episodic memory are replayed every time results in increased learning time when a new object category is encountered. As discussed above using a selective replay of neurons in the episodic memory during memory replay can address this issue. If all the above-mentioned limitations are addressed, the proposed learning system with limited storage complexity and considerable learning time. This may serve as a robust technique for the robotic system which will be aware of the objects in its environment and with object grasping skills it can perform different types of manipulation tasks towards lifelong learning, similar to humans. This type of hybrid continual learning system can be further adapted to develop the complete autonomous robotic systems by improving the model knowledge to address the sensory and motor functions like speech recognition, localization, and robot motion control estimators.

Bibliography

[1] Sumit Saha, "A comprehensive guide to convolutional neural networks," 2021. [Online; accessed 25-August-2021].

[2] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2021.

[3] S. Marsland, J. Shapiro, and U. Nehmzow, "A self-organising network that grows when required," *Neural Networks*, vol. 15, no. 8, pp. 1041–1058, 2002.

[4] J. O. Vrielink and H. Kasaei, "Learning to grasp objects in highly cluttered environments using deep convolutional neural networks," 2021.

[5] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[6] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[7] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017.

[8] M. Mermillod, A. Bugaiska, and P. BONIN, "The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects," *Frontiers in Psychology*, vol. 4, p. 504, 2013.

[9] A. Cangelosi and M. Schlesinger, *Developmental robotics: From babies to robots*. MIT press, 2015.

[10] J. Tani, *Exploring robotic minds: actions, symbols, and consciousness as self-organizing dynamic phenomena*. Oxford University Press, 2016.

[11] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 25–46, 1995.

[12] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[13] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Networks*, vol. 116, pp. 56–73, 2019.

[14] J. D. Power and B. L. Schlaggar, "Neural plasticity across the lifespan," *Wiley Interdisciplinary Reviews: Developmental Biology*, vol. 6, no. 1, p. e216, 2017.

[15] D. J. Lewkowicz, "Early experience and multisensory perceptual narrowing," *Developmental psychobiology*, vol. 56, no. 2, pp. 292–315, 2014.

[16] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? complementary learning systems theory updated," *Trends in cognitive sciences*, vol. 20, no. 7, pp. 512–534, 2016.

[17] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.," *Psychological review*, vol. 102, no. 3, p. 419, 1995.

[18] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[19] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[20] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[21] G. I. Parisi, J. Tani, C. Weber, and S. Wermter, "Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization," *Frontiers in Neurorobotics*, vol. 12, p. 78, 2018.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] P. Marius, V. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, 07 2009.

[24] R. Lasri, "Clustering and classification using a self-organizing map: The main flaw and the improvement perspectives," in *2016 SAI Computing Conference (SAI)*, pp. 1315–1318, 2016.

[25] Wikipedia contributors, "Gaussian noise — Wikipedia, the free encyclopedia," 2021. [Online; accessed 26-August-2021].

[26] Wikipedia contributors, "Voxel — Wikipedia, the free encyclopedia," 2021. [Online; accessed 26-August-2021].

[27] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[28] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[29] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*, pp. 3987–3995, PMLR, 2017.

[30] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[31] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.

[32] A. Gepperth and C. Karaoguz, "A bio-inspired incremental learning architecture for applied perceptual problems," *Cognitive Computation*, vol. 8, no. 5, pp. 924–934, 2016.

[33] J. L. Part and O. Lemon, "Incremental on-line learning of object classes using a combination of self-organizing incremental neural networks and deep convolutional neural networks," in *Workshop on Bio-inspired Social Robot Learning in Home Scenarios (IROS), Daejeon, Korea*, 2016.

[34] G. I. Parisi, J. Tani, C. Weber, and S. Wermter, "Lifelong learning of human actions with deep neural network self-organization," *Neural Networks*, vol. 96, pp. 137–149, 2017.

[35] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1–1, 2021.

[36] S. Jain and H. Kasaei, "3d_den: Open-ended 3d object recognition using dynamically expandable networks," 2021.

[37] J. L. Part and O. Lemon, "Incremental online learning of objects for robots operating in real environments," in *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 304–310, IEEE, 2017.

[38] J. C. Principe, J.-M. Kuo, and S. Celebi, "An analysis of the gamma memory in dynamic neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 2, pp. 331–337, 1994.

[39] L. Erculiani, F. Giunchiglia, and A. Passerini, "Continual egocentric object recognition," 2020.

[40] S. H. Kasaei, "Look further to recognize better: Learning shared topics and category-specific dictionaries for open-ended 3d object recognition," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5438–5443, IEEE, 2019.

[41] S. H. Kasaei, M. Oliveira, G. H. Lim, L. S. Lopes, and A. M. Tomé, "Towards lifelong assistive robotics: A tight coupling between object perception and manipulation," *Neurocomputing*, vol. 291, pp. 151–166, 2018.

[42] S. H. o. Kasaei, "Orthographicnet: A deep transfer learning approach for 3d object recognition in open-ended domains," *IEEE/ASME Transactions on Mechatronics*, pp. 1–1, 2020.

[43] H. Ayoobi, H. Kasaei, M. Cao, R. Verbrugge, and B. Verheij, "Local-hdp: Interactive open-ended 3d object categorization in real-time robotic scenarios," 2021.

[44] S. H. Kasaei, L. S. Lopes, and A. M. Tomé, "Local-lda: Open-ended learning of latent topics for 3d object recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2567–2580, 2020.

[45] S. Ghadai, X. Y. Lee, A. Balu, S. Sarkar, and A. Krishnamurthy, "Multi-level 3d cnn for learning multi-scale spatial features," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1152–1156, 2019.

[46] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," 2016.

[47] D. Morrison, P. Corke, and J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," 2018.

[48] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3d object grasp synthesis algorithms," *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 326–336, 2012. Autonomous Grasping.

[49] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg, "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1957–1964, 2016.

[50] J. Yu, K. Weng, G. Liang, and G. Xie, "A vision-based robotic grasping system using deep learning for 3d object recognition and pose estimation," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1175–1180, 2013.

[51] D. Watkins-Valls, J. Varley, and P. Allen, "Multi-modal geometric learning for grasping and manipulation," 2019.

[52] S. H. Kasaei, N. Shafii, L. S. Lopes, and A. M. Tomé, "Interactive open-ended object, affordance and grasp learning for robotic manipulation," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3747–3753, 2019.

[53] 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)*, pp. 2895–2900, IEEE, 2016.

[54] Y. Li, L. Schomaker, and S. H. Kasaei, "Learning to grasp 3d objects using deep residual u-nets," 2020.

[55] H. Kasaei and M. Kasaei, "Mvgrasp: Real-time multi-view 3d object grasping in highly cluttered environments," 2021.

[56] S. Kumra, S. Joshi, and F. Sahin, "Antipodal robotic grasping using generative residual convolutional neural network," 2021.

[57] H. Kasaei, S. Luo, R. Sasso, and M. Kasaei, "Simultaneous multi-view object recognition and grasping in open-ended domains," 2021.

[58] A. Depierre, E. Dellandréa, and L. Chen, "Jacquard: A large scale dataset for robotic grasp detection," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3511–3516, 2018.

[59] M. A. Uy, Q.-H. Pham, B.-S. Hua, T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[60] V. Lomonaco and D. Maltoni, "Core50: a new dataset and benchmark for continuous object recognition," in *Proceedings of the 1st Annual Conference on Robot Learning* (S. Levine, V. Vanhoucke, and K. Goldberg, eds.), vol. 78 of *Proceedings of Machine Learning Research*, pp. 17–26, PMLR, 13–15 Nov 2017.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[62] I. Dokmanic, R. Parhizkar, J. Ranieri, and M. Vetterli, "Euclidean distance matrices: Essential theory, algorithms, and applications," *IEEE Signal Processing Magazine*, vol. 32, p. 12–30, Nov 2015.

[63] Wikipedia contributors, "Euclidean distance — Wikipedia, the free encyclopedia," 2021. [Online; accessed 23-August-2021].

[64] S. Craw, *Manhattan Distance*, pp. 790–791. Boston, MA: Springer US, 2017.

[65] Wikipedia contributors, "Minkowski distance — Wikipedia, the free encyclopedia," 2021. [Online; accessed 23-August-2021].

[66] *Mahalanobis Distance*, pp. 325–326. New York, NY: Springer New York, 2008.

[67] B. Li and L. Han, "Distance weighted cosine similarity measure for text classification," in *Intelligent Data Engineering and Automated Learning – IDEAL 2013* (H. Yin, K. Tang, Y. Gao, F. Klawonn, M. Lee, T. Weise, B. Li, and X. Yao, eds.), (Berlin, Heidelberg), pp. 611–618, Springer Berlin Heidelberg, 2013.

[68] Y. Jiang, S. Moseson, and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3304–3311, 2011.