university of groningen / faculty of science and engineering

Deep Representation Learning for 3D Object Recognition in Open-Ended Domains

Bachelor's Project Thesis

Vlad Cosmin Iftime, s3426394, v.c.iftime@student.rug.nl,
Supervisor: Dr. Hamidreza Kasaei, hamidreza.kasaei@rug.nl
Department of Artificial Intelligence

Abstract: Open-ended category learning refers to the act of learning new object categories, after the training phase, without forgetting the previously learned categories. Due to the limited number of training data and also the difficulty in reprogramming the entire architecture of the system when a new category is presented, open-ended category learning is a solution that still needs improvement in service robotics. To that end, this project analyzes the effects of four different autoencoder neural networks on the representation learning part of the OrtographicNet, a network designed by H. Kasaei to tackle 3D object recognition in open-ended domains. The four types of autoencoders analyzed in this project are a simple dense autoencoder, a convolutional autoencoder, a variational autoencoder, and an adversarial autoencoder. The autoencoders were first trained to provide a unique object feature representation using a self-supervised representation learning approach. The feature representation obtained using only the encoder part of the autoencoders was then used for both the recognition and learning processes. After exhaustive testing using different recognition algorithms parameters, similarity measures distances, and pooling functions, the best autoencoder model for learning and recognizing 3D objects in open-ended domains is the variational autoencoder, due to its prior Gaussian distribution that forces the latent space of the autoencoder to obtain disentangled features.

1 Introduction

Three dimensional (3D) object recognition has been one of the most attractive research topics in robotics, computer vision, and deep learning communities for more than half of the decade [1], mainly because of its applications in autonomous cars, human-machine interaction, bio-metrics, and entertainment. In terms of human-machine interaction, one of the ever-evolving fields, in which 3D object recognition is considered paramount, is service robotics. This branch of robotics focuses on robots aimed at performing tasks for humans (e.g., cooking, moving objects, delivery). Although huge leaps have been made into the development of such robots in the last year, one of the biggest hurdles that still plagues both 3D object recognition and service robotics is robustness. In this work, robustness refers to a system's ability to perform when presented with new conditions.

One task where robustness is an essential ability in service robots is category learning. While it is true that progress has been made in this area, there are still some problems where improvements have to be made. One proposed solution is *open-ended category learning* (OECL). OECL refers to a subject's ability to sequentially learn new object-categories without forgetting the previously learned categories [2][3]. This approach represents a form of continual learning [3], an effective machine learning paradigm for robotic agents.

Object categorization is a task quite mundane for humans. For example, we can learn what a wild animal (e.g., a tiger) looks like after receiving a small number of pictures of the animal. We can also learn this type of information without forgetting how other animals look. On the other hand, presenting a service robot with a never-seen-before object results in failure and retraining the robot's architecture. The solution to this kind of problem is evi-

dent; the robot needs to be able to learn new object categories while operating. Ideally, it should be able to do it without forgetting previously learned categories, and it should be able to update its knowledge with feedback from a human teacher.

From the example given above, we can already contour how the robot's learning system should look. First of all, the system should be able to learn new categories while online. This solves a big problem for the architect of the system by removing the necessity for a high number of training examples. The robot should also be able to adjust the representation that it learned for a category when a new instance is taught; thus increasing its overall robustness. Finally, a teacher should supervise the learning system by presenting new instances of categories and by providing feedback. We simulate a supervised teaching protocol in open-domains using the system developed by Kasaei et al. [4], which successfully replicates a simple interaction between a robot and a teacher.

Several systems have been developed to tackle the problem of OECL [5] [6]. In this paper, we focus on the system proposed by H. Kasei called OrthographicNet [5]. More specifically, this work focuses on the *Object representation* and the *Object category learning and recognition* part of the OrthographicNet and tries to improve its accuracy and robustness by replacing the convolutional neural networks with four types of autoencoders.

For this work, we keep the open-ended object representation pipeline from the OrthographicNet. This pipeline feeds the orthographic projection from three different views of an object (top, side, and front) to three identical CNNs that obtain a view-wise in-depth feature of each view. These views are combined to create a feature vector of the object, used in the system's OECL part.

1.1 Our approach

Autoencoders is a term used to describe a simple artificial neural network that tries to transform a given input into an identical output. The majority of autoencoders, including those discussed in this paper, are built based on the encoder-decoder paradigm. The input is first fed through the encoder to create a lower dimensionality representation, also known as a *latent layer* or simply *code*. The decoder then attempts to obtain the original image from the latent layer. Due to the lack of labels that the input requires, this sort of network offers the distinct advantage, over classic CNNs, of performing self-supervised representation learning.

Self-supervised representation learning represents a popular topic in machine learning and especially in robotics. The benefit of this type of learning is the ability to scale to large amounts of unlabelled data in a lifelong learning manner and improve performance by reducing dataset bias. Consequently, we seek to use the latent layer representation of an autoencoder network, trained using self-supervised representation learning, to optimize the supervised category learning that OrthographicNet proposes. More specifically, this work looks at four different types of autoencoders. In Fig. 1.1, the reader can find an example for each autoencoder architecture.

- **Dense autoencoders** In this autoencoder type, both the encoder and the decoder are constructed out of fully connected layers that increase/decrease in size as they get closer/further from the latent layer, i.e., code.
- **Convolutional autoencoders (CAE)** Proposed by Masci et al. [7] this type of autoencoder is similar to the dense autoencoder, but it replaces the fully connected layers with convolutional layers and max-pooling layers in between. As is the case with the dense autoencoders, the layers sizes increases/decreases as they get closer/further from the latent layer, i.e., code.
- **Variational autoencoders (VAE)** This type of autoencoder has a more complex architecture than the previous two, but the core still follows the encoder-decoder paradigm. On top of this paradigm, VAEs force the latent space into a Gaussian distribution due to the Gaussian sampling done after the encoding combined with a reparametrization term [8]. While VAEs have a broader usage than latent representation learning, more specifically as a generative model, it goes beyond this paper's scope to discuss such usage. We thus refer the reader to other works such as Walker et al. [9], where this topic is addressed in more detail. The encoder and decoder follow the same construction as the CAE.

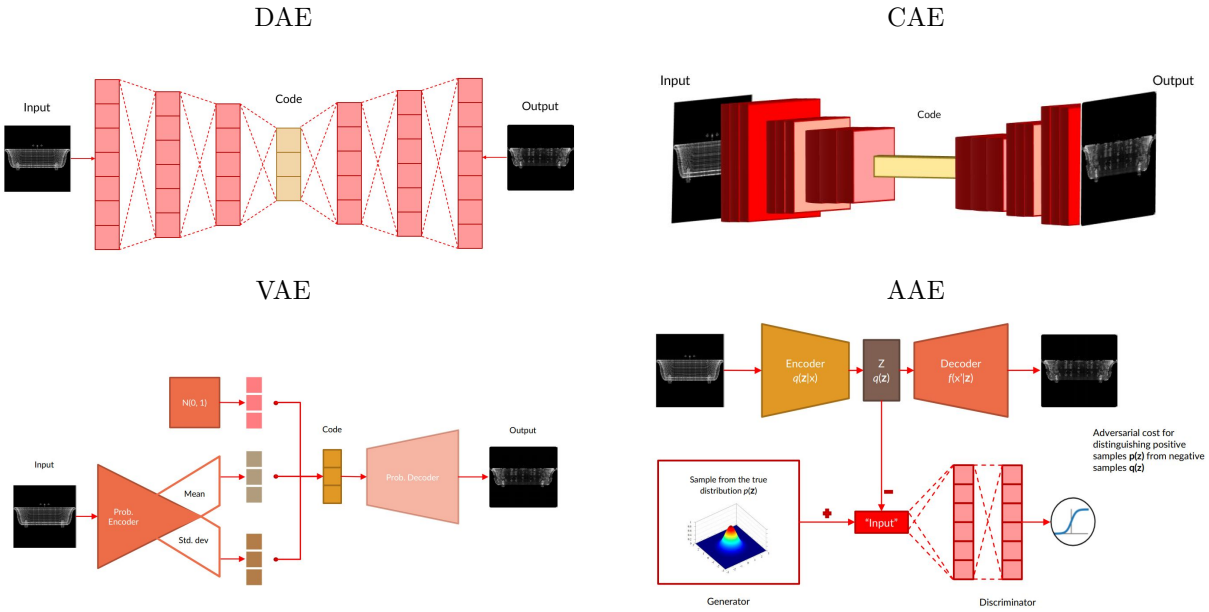

Figure 1.1: Architecture of each type of autoencoder.

- **Adversarial autoencoders (AAE)** These are the most recent type of autoencoders [10]. They use a generative adversarial network (i.e., GAN) [11] to perform variational inference to match the latent space distribution produced by the encoder with one of the data. As expected, this type of autoencoder can be used as a generative model, but this goes beyond this paper's scope. The VAE heavily inspires this model, but instead of using a regularization term in the loss function to force the distribution of the latent vector, it uses an adversarial network [10]. As with the VAEs, the encoder and decoder have the same layer construction as a CAE. On the other hand, the Discriminator part of the model is comprised of fully connected layers.

Their performance is compared based on the following four metrics: *how fast does it learn?, how much does it learn?, how well does it learn?, how much memory does it take?*. The way each of this measurements are calculated is explained in Section 4.3.

Each type of autoencoder is compared against each other and against the original OrthographicNet in order to answer the research question:

Which autoencoder models perform best for learning and recognizing 3D objects in open-ended domains?.

The rest of the paper is structured as follows: Section 2 describes related work in terms of OECL and autoencoders; Section 3 describes our implementation of the four autoencoder models and the experimental setups; Section 4 presents and discusses the results of the experimental setups; Section 5 draws conclusions based on the results of Section 4 to answers the research question, and also presents what future research could be done.

2 Related work

Our research focuses on two topics, namely *open-ended category learning* and *autoencoders*.

2.1 Open-ended category learning

The idea of a robot able to adapt to its environment has been around as far back as the late 1990s. T. Ziemke [12] proposed the use of recurrent neural networks (RNNs) to create adaptive behavior for robots. The project results showed that RNNs could provide a middle ground between utterly re-

active behavior and an explicit model of the world. The task that the robot was able to adapt to was an obstacle avoidance task in an increasingly complex environment. While the robot in [12] was able to adapt to the environment, little learning was taking place; this is mainly due to the constraints of technology available at the time, i.e., RNNs.

More recently, continual learning emerged as a paradigm that aims to solve the adaptability and robustness challenges of robotics [3][13]. One specific area where this type of learning has made great advances and where service robots can be improved is *object recognition/categorization.* Three approaches that have made progress into these tasks are [14], [15], and [4][16]. The latter is of great importance to this paper. Li et al. [15] propose an approach, called *learning without forgetting.* The idea is self-explanatory; they propose a system capable of learning new classification tasks while maintaining old task's capabilities. This type of ability is, of course, a very desirable aspect for a service robot. As mentioned in the introduction, a robot should learn what represents a table, without forgetting how a cup or a plate look. The proposed approach focuses on solving the problem of *catastrophic forgetting* [17], [18] by learning parameters that are specific for new tasks while at the same time keeping the ones discriminative for the old tasks.

Gidaris et al. [14] follows the same strategy of learning without forgetting. Their work tries to improve onto the classical few-shot learning approach, [19] by learning new categories without forgetting the previously learned ones. They do this by creating a weight generator that helps in recognizing novel categories and keeping a high performance for the categories used for training.

Both the approaches of Li et al. [15] and Gidaris et al. [14] can produce competitive results in terms of recognition and categorization, but their major pitfall is that they are unable to operate online. They cannot work if the data is presented in a stream, and the model needs to perform a task while also training. Kasaei et al. [4] solve this problem by using OECL, an approach that combines the continual learning aspect of remembering previously learned tasks with cumulative learning [20]. The approach of Kasaei et al. [4] focuses on *instance based category learning* [21], where the instances represent each category that contains them.

2.2 Autoencoders

Autoencoders are a type of artificial neural network i.e., ANN that was introduced in 1986 by Hinton et al. [22]. Hinton et al. [22] proposed these networks as a solution to the problem of backpropagation without the need of a teacher. Thus they got to be in the center stage of the unsupervised machine learning paradigm [23]. They have also been used in forms of representation learning [24] due to their ability to create a latent representation that encapsulates the main features of data. In this regard, an autoencoder can be taught as a non-linear principal component analysis, i.e. PCA, alternative [25]. The latter aspect of autoencoders are why this type of ANN is the focus of our research. Thus by taking advantage of the representation learning capabilities of autoencoders, we produce competitive results in OECL.

Convolutional neural networks are known to be good discriminative neural networks [25] and thus have been used in countless projects regarding computer vision tasks such as recognition [26], [27] and 3D object classification [28]. On the other hand, one of the biggest drawbacks of these kinds of networks is their rigidity in introducing new categories. One way of avoiding this hurdle is by using CNNs pre-trained on large datasets [29]. These pre-trained models can offer good features extraction, and as such, they have been used by Kasaei et al. [5].

On the other hand, autoencoders are known more as generative models [25], but studies such as Vincent et al. [30] have shown that using the latent space that they generate we can extract descriptive features from our input. These characteristic features can remove ambiguity from different objects that belong to the same category and can be used in combination with instance-based learning, where these very descriptive vectors can be combined to represent instances of each category.

3 Methods

The experimental setup that answers the proposed research question of this paper consists of three distinct phases. The *training phase*, the *offline phase* and the *online phase.*

3.1 Training phase

Self-supervised representation learning is one of the main benefits that autoencoders provide. It offers the distinct advantage that there is no need for labels on the data. That is why the autoencoders are first trained on a set of images to make sure that they can extract meaningful features. This training phase aims to prepare the encoder of the autoencoders to be used in a future task, as a form of transfer learning [31].

In this first phase i.e., the training phase, we implement the four types of autoencoders, and for each, we determine their optimal architecture and loss function. To guide our model construction, we use the findings of [32] as well as exhaustive trial and error.

One crucial aspect for all autoencoders, which is paramount to consider when deciding the architecture, is the *dimensionality of the latent space*. This parameter has a significant effect on the quality of the images that the autoencoder can reconstruct from it. On the one hand, a low-dimensional space is memory and time efficient during the training process, but it might not capture the essential features of the input. On the other hand, a high-dimensional space can lead to over-fitting while also requiring more time and memory to train. In order to analyze this effect on the models, we look at an increasingly larger latent space. Thus, the values used for analyzing the latent dimensionality effect are *2, 8, 32, 64*.

Lastly, in this phase, the other two parameters that we look at are the batch-size and the number of epochs that we train each model. There are extensive research papers done into how these training parameters affect the model's performance, and they served as a guide for what values we chose. Another method that these values are usually determined through trial and error. For this reason, no specific values have been predetermined for testing.

For all the models the layer arrangements are guided by the models in Section 1.2. All the autoencoders are programmed in Python 2.7 using the Keras framework.

3.1.1 Dense autoencoder

The dense autoencoder is built out of an encoder with four layers. Each layer's sizes depend on the size of the latent space, i.e., s_l and is half the size of the previous layer. Thus the first layer has a size of $16 \times s_l$, and the last layer has a size of $2 \times s_l$. The decoder is the mirror of the encoder.

The loss function we use for this autoencoder is a *mean squared error*:

$$(X - Y)^2 \tag{3.1}$$

where X is the input, and Y is the output.

3.1.2 Convolutional autoencoder

For the convolutional autoencoder, we choose to have three convolutional layers. The first one has 16 filters, while the other two have 8. The kernel size for each of them is three by three.

There is a max-pooling layer between each convolutional layer that makes the input from the previous layer half as big. Thus the final max-pooling layer will output the size of the latent space. The decoder uses upsampling layers to double the size of the inputs they receive.

The loss function that we use for this model is *Binary Cross-entropy* i.e.

$$-1(X_i \times log(Y_i) + (1 - X_i) \times log(1 - Y_i)) \tag{3.2}$$

where X_i and Y_i represent pixels from the input and the output respectively.

3.1.3 Variational autoencoder

The variational autoencoder has both the encoder and decoder part identical to the convolutional autoencoder with the following exception. The encoder contains three different layers right before the latent space. As in Fig. 1.1 (bottom left), we will have three other layers. A *mean layer (μ_x)* that represents the mean of the input distribution, a *covariance layer (σ_x)* that represents the covariance of the input distribution and finally a *sampling layer (ζ)* that samples a point from a normal distribution i.e. a distribution with $\mathcal{N}(0, \mathbf{I})$. The sampling for this layer is done using the reparameterization trick [8] to ensure that the backpropagation has the desirable effects.

The latent space, in this model, represents a distribution and is calculated in the following way:

$$z = \sigma_x \zeta + \mu_x \tag{3.3}$$

The loss function for this model uses the following formula:

$$\mathcal{L}(\phi,\theta,x)=D_{KL}(q_\phi(h|x)\|p_\theta(h))-\mathbb{E}_{q_\phi(h|x)}(logp_\theta(x|h)) \quad (3.4)$$

ϕ represents the parameters for the encoder while θ represents the ones for the decoder; the first term of the function is called the regularizer term, ensuring that the latent distribution is similar to the sampled one. For this, the Kullback–Leibler, i.e. KL, divergence is used to calculate the difference between the two distributions. The second term is the reconstruction likelihood, and it ensures that the output is as similar to the input as possible, it is implemented as a binary-cross-entropy function.

A last note about the implementation of this model: in our experimental setup, we experienced *posterior collapse*, where the autoencoder reaches a point in the learning process where they ignore the latent variable and thus generate only trivial solutions [33]. To avoid it, we implement a slightly modified version of the original loss function proposed by Bowman et al. [34]. In this loss function, the regularizer term is annealed and thus introduced gradually into the function. In this way, the model will first learn to reconstruct the input precisely before it starts forcing the distribution.

3.1.4 Adversarial autoencoder

As stated in the introduction, this autoencoder model is heavily inspired by the variational autoencoder [10]. Thus the encoder and decoder follow the same construction as the VAE and the CAE.

In addition to the encoder's classical construction, we also build a discriminator network built out of three fully connected layers. The first layer receives input from a vector that is the size of the latent space, and the last layer outputs either a negative or a positive response. The layers have the following sizes from first to last: 512, 256, 1. The input for this network is either a latent vector generated by the encoder or a sampled vector. The sampling is done similarly to the VAE. We sample using a reparameterization trick from a normal distribution i.e. $\mathcal{N}(0, \mathbf{I})$.

At each training epoch, this model trains both the generative network, which is represented by the encoder-decoder network and the discriminator network. The generative network has the same goal as a simple autoencoder to ensure the output is similar to the input. As such, it uses a *mean squared error*, i.e. MSE, as a loss function. The discriminator network tries to differentiate between the true distribution i.e., $\mathcal{N}(0, \mathbf{I})$ and the latent representation from the encoder. By considering the latter as fake, we force the latent vector's distribution to be similar to the true distribution until the discriminator can no longer tell the difference.

3.1.5 Data

The data that we are using in the training phase comes from the ModelNet10 dataset [35]. While the data comes from this dataset, we use the altered version that fits the OrthographicNet pipeline proposed by H. Kasaei [5] where orthographic projections of household items represent the data. The term orthographic projection is a technical term commonly used by engineers and other professionals who use technical drawings. From this point on, in this paper, the ModelNet10 dataset will refer to the one proposed by H. Kasaei and not the original one containing CAD models.

ModelNet10 contains 4899 household items divided into ten classes: bathtub, bed, chair, desk, dresser, monitor, nightstand, sofa, table, toilet. It can be observed that the dataset is small, especially when compared with other image datasets such as ImageNet [36]. A smaller dataset represents an advantage because it offers the possibility for more extensive experiments. Another aspect of the dataset is the interclass variation of the ten classes. This variation is a paramount aspect for OECL that ModelNet10 offers.

Each class contains representations of objects corresponding to the class. There are three main orthographic views of each object: front, top, and right-side. The authors of OrthographicNet did not consider the other three views due to them being mirror images of the other three. One picture represents one of the three main views; thus, there are three pictures per object. Each picture is 100 by 100 pixels in JPEG format, and all the pixels are normalized to have values between 0 and 1.

3.1.6 Pre-processing

One way of enhancing the model's overall performance is by making sure that the data used to train it is adequate and not undermined by noise.

To that end, the data used to train the autoencoders was first passed through two different pre-processing functions. The pre-processing is predominantly done to reduce the noise in the picture and thus increase the model's accuracy.

The two different pre-processing function that are implemented are: *erosion* and *dilation*. Since there is no objective way of determining which one would improve the performance of the model, the entire training data set is passed to each function separately, thus resulting in two different dataset each corresponding to a pre-processing function. Each autoencoder is trained using both datasets to see which combination of autoencoder-pre-processing function would result in the best overall model. The dilation and erosion work on the orthographic representation because they are black and white. As such, orthographic projections are prone to white noise due to the sensors, and both these functions propose a solution to this. Erosion works by removing white noise from the object's boundaries in the picture by sliding a kernel over the image and keeping a pixel only if all the other under the kernel are also white. Because the erosion process might end up making the orthographic representation of the object too small, the dilation function is used to 'increase' the object back closer to its original form, with the added benefit of removing the noise.

Dilation works similarly to erosion. Thus the dilation function is implemented in this project by first eroding the picture and then dilating it. The two functions are implemented using the OpenCV's Python extensions. To visualize how these two functions perform on the ModelNet10 data set we refer the reader to Fig. 3.1.

The final pre-processing step is to shrink the images to a 64 by 64 pixel size.

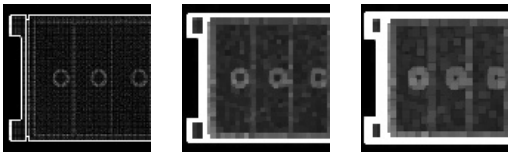

Figure 3.1: (*left*) The original orthographic projection; (*middle*) The erosion function applied to the original; (*right*) The dilation function applied to the original.

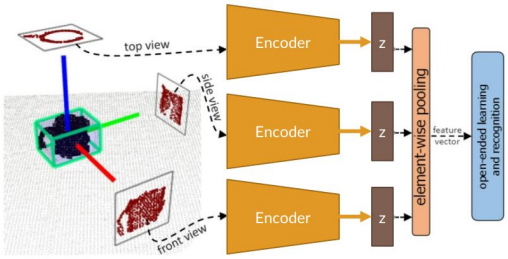

Figure 3.2: The OrthographicNet pipeline where the CNNs are replaced with the trained encoders that produce a latent representations of the input images, i.e. z. The image is adapted from H. Kasaei [5]

3.2 Offline phase

While OrthographicNet offers a solution to the online nature of OECL and continual learning, we first have to integrate the autoencoders with the most performant weights, developed in the previous phase, into OrthographicNet and evaluate them. The integration is an application of transfer learning [31].

To that end, we conduct a series of experiments. For each autoencoder model, we find the most suitable combination of object representation characteristics, similarity measures, and pooling function. These combinations are then passed onto the final phase of the experimental setup.

The type of experiment used for finding the most suitable combination for each type of autoencoder is K-fold cross-validation. The reason why we choose this type of cross-validation over the other types is due to its ability to offer a less biased performance assessment for each combination of an autoencoder and pre-processing functions applied to it. The same dataset is used due to its inter-class variation and its size. These aspects allow for a high number of experiments to be done in a short amount of time.

3.2.1 OrthographicNet integration

As mentioned in the introduction, we replace the CNN in the original OrthographicNet proposed by H. Kasaei [5] with the trained encoders from the autoencoders developed in the previous phase.

Fig. 3.2 shows how the open-ended object recognition pipeline looks like when the encoder is at-

tached. We first test different resolutions for the three projected views. The values that we look at are *32, 64, 96*. These values were chosen following the size of the latent space of the autoencoder model spaces.

After the orthographic representations are fed to the encoders, they each produce a latent representation of these representations. The three latent representations are then pooled to create a single feature vector. The two pooling functions that we test are *max pooling* and *average pooling*, which are commonly used throughout the literature. The formulas for the two pooling functions are the following, 3.5 and 3.6, where *f = feature vector; x = first latent space vector; y = second latent space vector; z = third latent space vector; i = position in vector.*

Max pooling formula

$$f_i = max(x_i, y_i, z_i) \tag{3.5}$$

Average pooling formula

$$f_i = avg(x_i, y_i, z_i) \tag{3.6}$$

These functions help create a feature vector that is invariant to small translations of the input. The creation of the feature vector marks the end of the object representation part of the pipeline. Finally, using the feature vector, the system learns and categorizes the original object.

The OECL part of the pipeline is done through the use of a K-nearest neighbor algorithm. This algorithm is well known in the machine learning field and is used for clustering objects based on their similarities. We test different neighborhood sizes for the algorithm to see which one would give competitive results. The neighborhood sizes that we use are *1, 3, 5, 7, 9*. It is considered good practice only to take odd values smaller than the number of categories in the dataset to avoid the possibility of a draw as much as possible.

In order to determine the distance from one object to another, we follow the study of S. Cha [37]. We use a total of 16 different distance functions from the ones proposed by S. Cha [37] :*KL-Divergance, SymmetricKL, Motyka, Divergence, Euclidean, Manhattan, Intersection, Cosine, Dice, Bhattacharyya, Sorensen, Canberra, Pearson, Neyman, Gower, Chi-squared*. These functions determine the distance between different feature vectors, while the K-nearest neighbor algorithm categorizes them based on the distance. We refer the reader to S. Cha [37] for a in depth description of each function.

In total we run *480* experiments for each model of autoencoder. The experiments represent all the combinations of $\#resolutions \times \#K\ values \times \#distance\ functions \times \#pooling\ functions = 3 \times 5 \times 16 \times 2 = 480$.

3.3 Online phase

A robot is capable of online learning if it can learn new pieces of information while operating without the need to reprogram its system. This is a useful feature that can help different sorts of service robots while operating on the field, where taking the robot offline to train it may be undesirable or even impossible(e.g., a robot working in the medical field or on the site of a construction). One approach to online learning is to have a human teacher present along the robot to provide feedback and new information.

This type of learning is a staple for OECL and continual learning and, as a result, also for this paper's scope. In this phase, the four OrthographicNet alternatives built using the four different autoencoders models are subjected to an *Open-ended evaluation* where we can determine the answer to the proposed research question: *Which autoencoder models perform best for learning and recognizing 3D objects in open-ended domains?.*

3.3.1 Simulated user

While the offline phase did provide insightful results into autoencoder's performance in terms of object representation learning and categorization, they are not suited for the evaluation of OECL. The reason being that they follow the train-test methodology. This two-stage approach overlooks the aspect of simultaneous recognition and learning and also requires full knowledge of the categories to be learned [16].

Thus, to correctly assess the autoencoder's performance in open-ended domains, we will use a simulated user protocol [38]. This protocol simulates the interaction of a human user with a robot over a number of interactions. The overall working of the protocol can be considered to be a train-test method that follows more closely to

how human teaching occurs. More specifically, a simulated-teacher interacts directly with the OrthographicNet system trough three tasks:

- **Teach**, the simulated-teacher presents a new object category to the system.
- **Ask**, the simulated-teacher presents an object to the system and asks for the category of said object.
- **Correct**, in case the system provides the wrong category when asked, the simulated-teacher will provide corrective feedback. During this task, the learning occurs.

The way online learning occurs is as follows [39]. The simulated-teacher continually monitors the global recognition accuracy of the system, which is computed as:

$$\textbf{Global accuracy} = \frac{\#correct\,predictions}{\#presented\,instances} \quad (3.7)$$

When this accuracy exceeds a given threshold, the simulated-teacher presents a new category to the system. The threshold is $\tau = 0.67$; this way, we ensure that the global accuracy is at least twice the error rate. While the system does not exceed the threshold, the simulated-teacher chooses new objects from the already presented categories and inquires about their category.

The experiments halt in one of two cases. The system cannot surpass the global accuracy in a given number of epochs (e.g., 100) the simulated-teach considers that the system is unable to learn any new categories and stops the experiment. The second case is when the system learns all the categories in the dataset. There is no reason to continue the protocol in such a case, and the experiment is stopped due to *lack of data*. This is the preferable case because it shows that the model is capable of learning even more.

3.3.2 Dataset

The dataset used for this phase of the experimental setup is the *Washington RGB-D Object dataset* [40]. This dataset is more extensive when compared to ModelNet10. It contains a total of 300 everyday household objects split into 51 categories. Since this is the last step of the experimental setup, and there are only four types of systems that we test, it makes sense to use a more comprehensive dataset to answer the research question.

4 Experimental results and discussion

For each phase described in the previous section, we perform a type of experiment. In this section, we look at the results of those experiments and analyze what they mean for our project and research question.

4.1 Training evaluation

In the training phase, we look to recognize the best architecture for each of the four types of autoencoders. The results of this evaluation are represented by *the size of the latent vector(i.e., code)*, *the number of epochs*, and *the size of the batches used to train the weights of the model*. This code sizes and training parameters provide the best weights for each autoencoder in the following two evaluations. It is worth noting that in this experimental set-up, we lastly evaluate which pre-processing function provides the medium for self-supervised representation learning.

Since the data for this section is unlabelled, we cannot calculate the accuracy to see which parameters will give the best results. Thus, we use the loss function for each type of autoencoder to see which parameters give the smallest loss between the input and the output.

Another aspect to note about the results of this phase is that because the loss functions are different, as noted in Section 3.1, we cannot directly compare the performances of the different autoencoders.

We first determine the size of the latent vector by running a classical train-test protocol with the same batch-size and number of epochs on increasingly larger latent space sizes. The results can be observed in Fig. 4.1. These results are averaged over 10 runs.

It can be observed that the relation between the value of the loss function at the end of the training phase and the size of the code is inversely proportional for the most part. This relation is valid

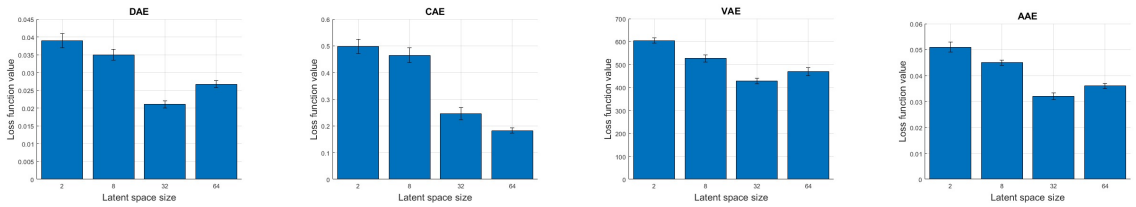

Figure 4.1: The loss function value in terms of the size of the latent vector for each model. The dense autoencoder (*first*); The convolutional autoencoder (*second*); The variational autoencoder (*third*); The adversarial autoencoder (*fourth*).

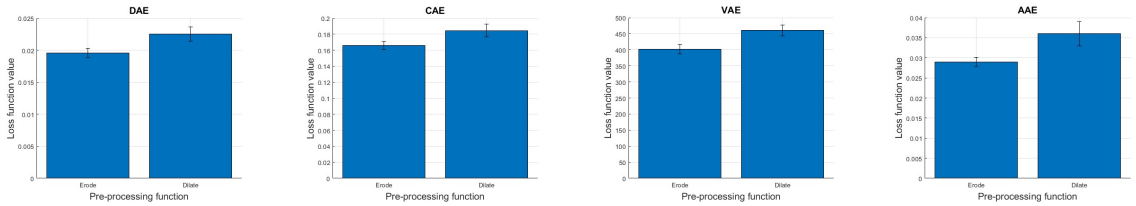

Figure 4.2: The loss function value in terms of the pre-processing function used on the input. The dense autoencoder (*first*); The convolutional autoencoder (*second*); The variational autoencoder (*third*); The adversarial autoencoder (*fourth*).

for the dense, variational, and adversarial autoencoders until the code reaches size 64. At this size, the value increases, which may show that the autoencoders start to overfit. The exception to this is the convolutional autoencoder. For this autoencoder, the loss for sizes 32 and 64 is similar. For all the other encoders, we continue our experimental set up with a latent space with size 32, while for the convolutional model, we use a size of 64.

Table 4.1 shows the results for the number of epochs and the size of the batches used to train the weights of the model. As we can see, the more complex the model's architecture is, the larger the number of epochs required to train it is. The adversarial autoencoder required the most significant number of epochs to train and the largest batch size. These parameters result from the fact that the adversarial model contains two networks that require training, the generator and the discriminator.

Table 4.1: The Number of Epochs and the Batch size that result in the lowest loss values for each autoencoder model

Autoencoder model	Number of epochs	Batch size
Dense	100	216
Convolutional	300	216
Variational	500	216
Adversarial	1500	512

Finally, the pre-processing functions are tested. We look at how the erode and dilate functions affect the loss functions of the determined autoencoder models. The results, averaged over ten runs, can be observed in Fig. 4.2. The erode function performed better in all models. The dilate function removed a lot more of the detail by increasing the size of the outer edges and other white sections, which led to the models overestimating those specific section's size. Thus, the results are worse when using dilate because a lot of the smaller details are lost, especially in the dense and adversarial autoencoders.

4.2 Offline evaluation

Taking the weights trained on the models from the previous phase, in this evaluation phase, we look at how the different *object representations*, *similarity measures* and *pooling functions* affect the Average Instance Accuracy(AIA) and the Average Class Accuracy(ACA). AIA is calculated as the percentage of correctly classified training instances, while ACA is calculated as the overall categorie's average accuracy.

Table 4.2: The results of the offline evaluation, where the best combination of parameters is chosen for each autoencoder model

Autoencoder model	Image resolution	Distance function	K	Pooling function	AIA	ACA
Dense	64×64	Euclidian	3	Average	0.8524	0.8990
Convolutional	64×64	Euclidian	3	Average	0.8524	0.9067
Variational	64×64	Euclidian	3	Average	0.8524	0.9125
Adversarial	32×32	Dice	3	Average	0.8502	0.8524

For each autoencoder model we have to test all combinations of 4 image resolutions, 5 K-values for the KNN algorithm, 16 distance functions, and 2 pooling functions, we have a total of 480 experiments for each model. This leaves us with a large number of experimental results. As such, we only look at the top result for each model. These are presented in Table 4.2.

It can be seen that the three out of the four models performed best when the resolution of the orthographic projections was 64×64, and the distance function was Euclidian. For the adversarial model, the most performant resolution was 32×32 with the Dice distance function. Since no model performed best with the resolution 96×96, it can be observed that a higher resolution is not guaranteed to offer better performance, even though it offers more detail. On the other hand, a too-small resolution lacks the detail necessary to extract useful features for the categorization. Thus an in-between resolution offers the best results for most of the models. The exception is, again, the adversarial autoencoder that performs best on the lower resolution. This divergence could be due to the discriminator being able to impose a better representation in the latent space.

For both the K-value used in the KNN algorithm and the pooling function, all the models perform best when K is 3 and when the pooling function is Average. These results make sense because a smaller neighborhood size can offer less ambiguity in categorization. Simultaneously, an average pooling function does not entirely ignore features from two of the views, which is what the max-pooling function does, which can help with the ambiguity by offering a more robust feature vector.

Finally, if we are to look at these offline evaluation's performances, we can see that the worst-performing model is the adversarial one, in both the average instance accuracy, a value of 0.8502, and the average class accuracy value of 0.8524. These poor results may come from poor optimization in the previous phase or from the overall implementation.

Looking at the other three models, we can see that they all perform equally well in AIA with a value of 0.8524. The variational model has the best ACA performance, with a value of 0.9125, followed closely by the convolutional model, with a value of 0.9067, and then by the dense model, with a value of 0.899.

It can be noticed that all the autoencoders perform well, with all values for both AIA and ACA above 0.85. This performance is plausibly a result of using the dataset to conduct the evaluation that we used in the training phase.

In order to better understand the results of this offline evaluation, we can look at the confusion matrices of the models from Table 4.2. The matrices are presented in Figure 4.3. We can observe that all models have a problem identifying the desk, 55.8% accuracy for the AAE, and 60.5% for the rest of the models. These accuracies show the ambiguity in the data set as this category is mistaken most frequently with the sofa category i.e., 20.9% of the time the AAE categorizes a desk as a sofa while the rest of the models do it 19.8% of the time. This confusion seems odd at first, but both objects are mainly represented by flat surfaces with different attachments. Other notable confusions are between desk and tables, and between nightstands and dressers. This confusion makes more sense as the objects themselves are quite similar and can be confused even by humans.

4.3 Online evaluation

In this final experimental evaluation, we use the simulated user protocol, described in Section 3.3.1, to analyze the performance of the four models ob-

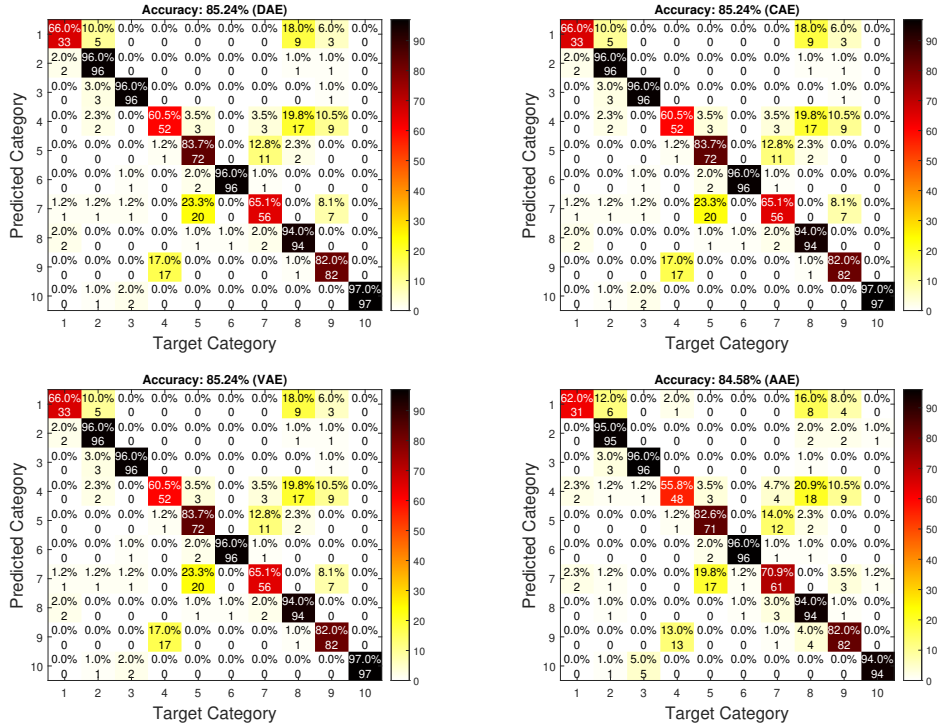

Figure 4.3: Confusion matrices showing how well each model performed in object recognition task. The darker diagonal cell is the better it was predicted by the model. Dense autoencoder *top-left*, Convolutional autoencoder (*top-right*), Variational autoencoder (*bottom-left*), Adversarial autoencoder (*bottom-right*).

tained in the previous evaluation in an open-ended domain. In this way, we will answer the proposed research question.

In the introduction, we mention that we assess the model's performance and subsequently answer the research question by looking at four metrics:

- *How fast does it learn?* For this we look at the number of Question / Correction Iterations i.e. **#QCI**
- *How much does it learn?* For this we look at the average number of categories that the model learned before the experiment stops i.e. **ALC**
- *How well does it learn?* In order to answer this question we look at the global accuracy of the classification i.e. **GCA**
- *How much memory does it take?* This is an evaluation of the memory efficiency of the model and for this we look at the average number of instances needed to learn a category i.e. **AIC**

It can be observed that for the metrics #QIC and AIC, a lower result is better as it shows that the model needs less time to complete the experiment and needs fewer objects from a category to learn said category as such it is better at generalizing.

To obtain these metrics, each autoencoder model is run through the simulated user protocol five times, and the averages of the four metrics is taken. We perform five experiments for each model because the order in which the categories are presented to the system may affect the performance. As we saw in the previous section, some objects are harder to categorise than others, and as such, if they were presented first, the experiment might end too fast.

The results of the open-ended evaluation can be found in Table 4.3.

Table 4.3: Summary of open-ended (online) evaluations

Model of autoencoder	#QIC	ALC	GCA	AIC
Dense	**984**	30.2	0.65	16.84
Convolutional	1822.2	**44**	0.67	16.21
Variational	1430.2	39.6	**0.68**	**14.33**
Adversarial	1338	34.6	0.66	15.6

The first aspect to note is that the relation between the #QIC and ALC scores is directly proportional, preferably inversely proportional. Thus, the faster the experiment finishes, the fewer categories the model is able to learn.

We can observe that in terms of learning speed, the best performing model is the dense autoencoder, with 984 iterations. This result is not entirely representative because, in all the cases, the experiment stops well before the model learns all the categories. We can observe that the model has the lowest ALC score of all models, only 30.2 categories learned. The second fastest model is the adversarial autoencoder, which needs 1338 iterations to learn 34.6 categories. The third fastest autoencoder model is the variational one, which needs 1430.2 iterations to learn 39.6 categories. Finally, the slowest model in terms of iterations, but the model that learns the most categories is the convolutional autoencoder, which learns 44 categories in 1822.2 iterations. Looking at the last two metrics, we can observe the model that performs the best in both metrics is the variational autoencoder, with GCA of 0.68 and an AIC of 14.33. The convolutional autoencoder performs second best in terms of GCA, with 0.67, and third-best in terms of AIC, with 16.21. The adversarial autoencoder performs third-best in terms of GCA , with 0.66 and second best, with a value of 15.60, in terms of AIC. Finally, the worst-performing autoencoder in both categories is the dense model with a value of 0.65348 for the GCA and 16.84 for the AIC.

We can note that the models that use a prior distribution to sample, the adversarial and variational autoencoder, generalize the best, as pointed by their AIC scores. They also perform quite similar in terms of global classification accuracy.

The odd aspect of this result is that there is a distinct difference in performance between the variational autoencoder and the adversarial autoencoder. It is odd because both models are guided by the same principle, as noted in the previous sections. The difference in performance can be attributed to the offline evaluation, too poor implementation or poor fine-tuning of the model.

To better understand and visualize the results presented above, we can look at the graphs from Fig. 4.4. These graphs show the results at the end of one of the simulated user protocols for the variational autoencoders. The first graph shows the evolution of protocol accuracy, i.e., the blue line over the number of iterations. When the simulated teacher introduces a new category (the red boxes), the accuracy drops suddenly. This drop occurs from the system not knowing what the category is due to the low number of instances it initially stores for the particular category. Another aspect worth pointing out from this graph is that as the model learns more categories, the speed with which it manages to pass the threshold again gets slower.

The middle graph presents the relation between global classification accuracy and the number of learned categories. As expected initially, the relation between the two is inversely proportional; the accuracy suffers a sharp drop as the model learns new categories. Eventually, the accuracy stabilizes and does not drop nor increase anymore.

The final graph shows how many instances the model stores for each category at the end of the protocol. The higher the number of instances, the harder it is for the model to learn that category. Some categories are innately ambiguous when looking at them just in terms of shape e.g., potato, tomato, bell pepper. These categories are quite variant in their shape, and thus it makes sense that they would be harder to learn by the model and require more instances to get them right. On the other hand, categories with more distinct shapes such as scissors, bowls, and plates require fewer instances to be learned.

For the graphs showing the performance of the other three models we refer the user to Appendix A.

5 Conclusions

In this section, we answer the research proposed in the introduction based on the previous section's results. We also look at how future research could

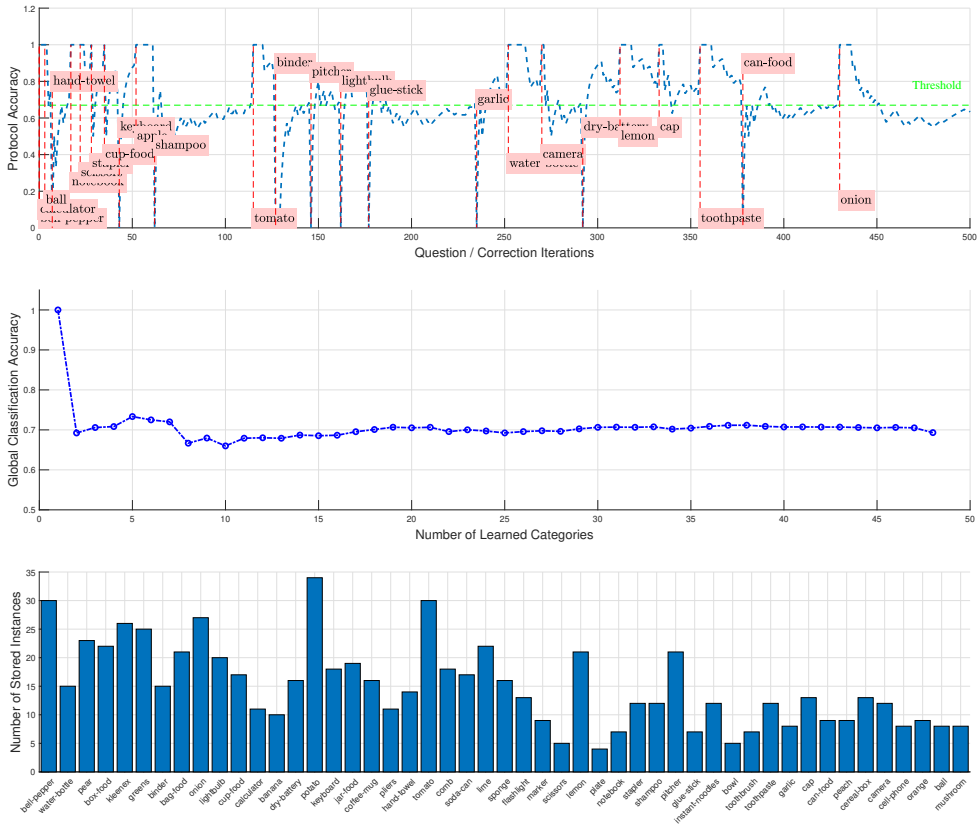

Figure 4.4: All the graphs are for the variational autoencoder model in the open-ended evaluation. The evolution of the learning protocol accuracy over the first 500 interactions (*top*); The global accuracy in terms of learned categories (*middle*); The number of instances that the model stored for each category (*bottom*)

build upon our project and what it should analyze.

5.1 Research question

In this project, we aimed to answer the following research question: *Which autoencoder models perform best for learning and recognizing 3D objects in open-ended domains?*. We performed three types of experiments using four different types of autoencoders integrated into the OrthographicNet system. The first two experiments helped us determine the optimal combination of parameters for each autoencoder. These combinations were tested in the last experimental setup, where we found the answer to the proposed research question. In this last evaluation, the variational autoencoder performed best regarding the generalization and overall accuracy, second-best in regards to the number of learned categories, and third-best in the speed of learning. Thus the autoencoder model that performs best for learning and recognizing 3D objects in open-ended domains is the **Variational Autoencoder**. This is expected because, unlike the other three models, the VAE provides a probability distribution of the latent space, which matches the data closer. Furthermore, due to the Gaussian priors that it samples in the latent space, the model learns disentangled factors, as pointed out by [41]. The rest of the models are ranked as follows based on the results in the open-ended evaluation: convolutional autoencoder, adversarial autoencoder and dense autoencoder.

5.2 Future research

While the VAE proved to be the most performant model out of the four that we tested, its results fall short of the ones obtained by H. Kasaei [5]. This shows that further research is needed to bring the autoencoders to the level of state of the art convolutional networks such as *VGG16* and *MobileNet-v2*. These are pretrained networks developed over a more extended period and were trained in ImageNet, the most diverse image dataset.

On the other hand, our autoencoders are trained on much smaller datasets, and due to time constraints, the fine-tuning is not performed at its fullest potential.

We propose that further research look into training the models on larger datasets and look at ways to improve the architecture of the models's trough fine-tuning. For example, other distributions, then the Gaussian distribution, could be tested for the sampling in both the variational autoencoder and the adversarial autoencoder. Also, even larger latent spaces could be tested and other pooling functions for the creation of the feature vector in OrthographicNet.

As a final point, due to time constraints and the global epidemic of COVID-19, real-time system demonstrations were not possible for our system. These experiments are fundamental to show the real-world applications of our project on service robots, and thus they should be tested in future research.

References

[1] A. Andreopoulos and J. K. Tsotsos, "50 years of object recognition: Directions forward," *Computer vision and image understanding*, vol. 117, no. 8, pp. 827–891, 2013.

[2] S. H. Kasaei, J. Sock, L. S. Lopes, A. M. Tomé, and T.-K. Kim, "Perceiving, learning, and recognizing 3d objects: An approach to cognitive service robots," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] A. S. D. M. D. F. N. D.-R. Timothée Lesort, Vincenzo Lomonaco, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020.

[4] S. H. Kasaei, A. M. Tomé, and L. S. Lopes, "Hierarchical object representation for open-ended object category learning and recognition," in *NIPS*, 2016.

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

[6] L. S. Lopes and A. Chauhan, "Open-ended category learning for language acquisition," *Connection Science*, vol. 20, pp. 277 – 297, 2008.

[7] J. Masci, U. Meier, D. C. Ciresan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *ICANN*, 2011.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014.

[9] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *ECCV*, 2016.

[10] A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow, "Adversarial autoencoders," *ArXiv*, vol. abs/1511.05644, 2015.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," *ArXiv*, vol. abs/1406.2661, 2014.

[12] T. Ziemke, "Remembering how to behave: Recurrent neural networks for adaptive robot behavior.," 1999.

[13] 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.

[14] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4367–4375, 2018.

[15] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 2935–2947, 2018.

[16] S. H. Kasaei, L. F. 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*, 2019.

[17] I. J. Goodfellow, M. Mirza, X. Da, A. C. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgeting in gradient-based neural networks," *CoRR*, vol. abs/1312.6211, 2014.

[18] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology of Learning and Motivation*, vol. 24, pp. 109–165, 1989.

[19] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang, "A closer look at few-shot classification," *ArXiv*, vol. abs/1904.04232, 2019.

[20] K. R. Thórisson, J. Bieger, X. Li, and P. Wang, "Cumulative learning," in *AGI*, 2019.

[21] D. W. Aha, D. F. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.

[22] G. H. D.E. Rumelhart and R. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing*, vol. 1, p. Foundations, 1986.

[23] D. Erhan, A. C. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?," *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, 2010.

[24] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504 – 507, 2006.

[25] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[26] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, pp. 1 – 62, 2020.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *CACM*, 2017.

[28] A. Sinha, J. Bai, and K. Ramani, "Deep learning 3D shape surfaces using geometry images," in *ECCV*, 2016.

[29] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 512–519, 2014.

[30] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *ICML '08*, 2008.

[31] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[32] D. Stathakis, "How many hidden layers and nodes?," *International Journal of Remote Sensing*, vol. 30, pp. 2133 – 2147, 2009.

[33] J. Lucas, G. Tucker, R. B. Grosse, and M. Norouzi, "Understanding posterior collapse in generative latent variable models," in *DGS@ICLR*, 2019.

[34] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating sentences from a continuous space," in *CoNLL*, 2016.

[35] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D shapenets: A deep representation for volumetric shapes," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1912–1920, 2015.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[37] S.-H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," 2007.

[38] S. H. Kasaei, M. Oliveira, G. H. Lim, L. S. Lopes, and A. M. Tomé, "Interactive open-ended learning for 3D object recognition: An approach and experiments," *Journal of Intelligent Robotic Systems*, vol. 80, pp. 537–553, 2015.

[39] S. H. Kasaei, L. S. Lopes, and A. M. Tomé, "Coping with context change in open-ended object recognition without explicit context information," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018.

[40] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset," *2011 IEEE International Conference on Robotics and Automation*, pp. 1817–1824, 2011.

[41] I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner, "Early visual concept learning with unsupervised deep learning," *ArXiv*, vol. abs/1606.05579, 2016.

A Appendix

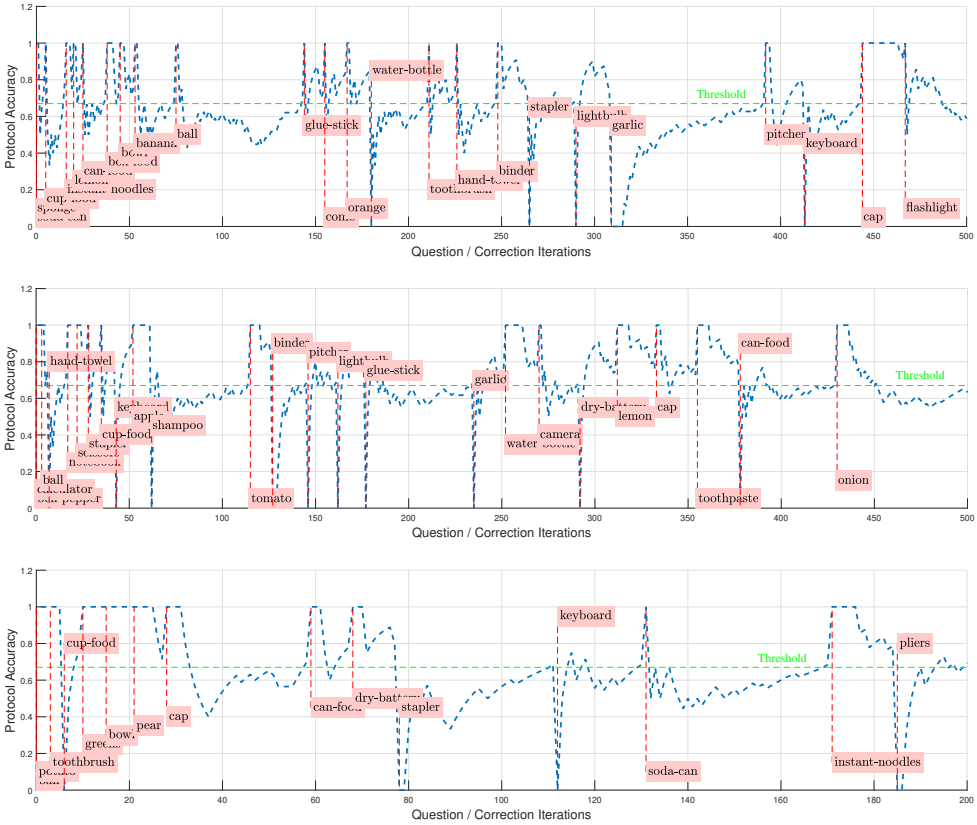

Figure A.1: The evolution of the learning protocol over the first 500 interactions for the open-ended evaluation. Dense autoencoder(*top*); Convolutional autoencoder(*middle*); Adversarial autoencoder(*bottom*)

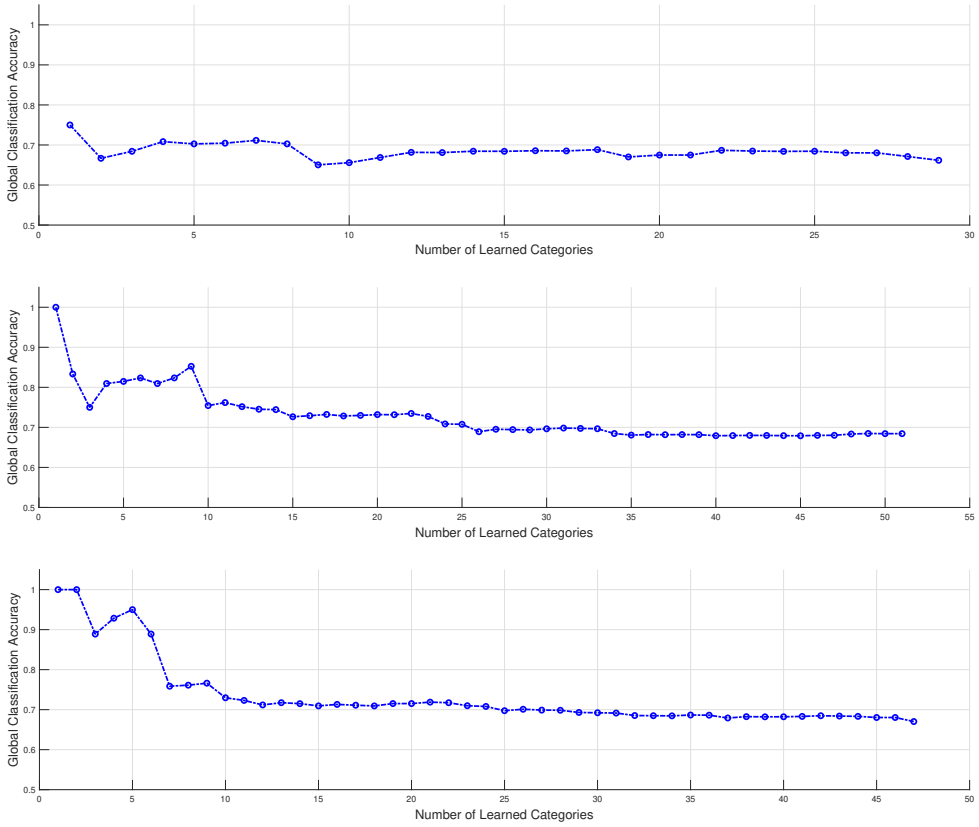

Figure A.2: The evolution of global accuracy as a function of learned categories. Dense autoencoder(*top*); Convolutional autoencoder(*middle*); Adversarial autoencoder(*bottom*)

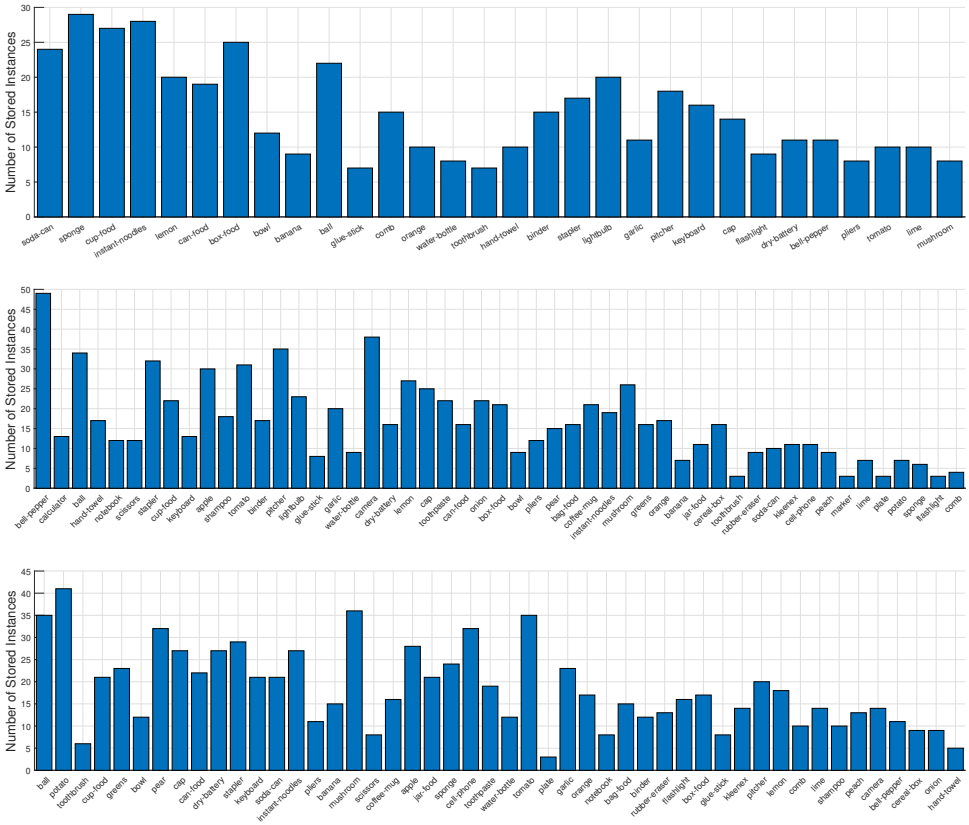

Figure A.3: The number of instances stored for each category. Dense autoencoder(*top*); Convolutional autoencoder(*middle*); Adversarial autoencoder(*bottom*)