
Open-Ended Learning Approaches for 3D Object Recognition

Hamidreza Kasaei
<http://www.ai.rug.nl/hkasaei>

Cognitive Robotics Course (2021-2021)
<https://rugcognitiverobotics.github.io>

Assignment overview

Cognitive science revealed that humans learn to recognize object categories ceaselessly over time. This ability allows them to adapt to new environments, by enhancing their knowledge from the accumulation of experiences and the conceptualization of new object categories. Taking this theory as an inspiration, we seek to create an interactive object recognition system that can learn 3D object categories in an open-ended fashion. In this project, “open-ended” implies that the set of categories to be learned is not known in advance. The training instances are extracted from on-line experiences of a robot, and thus become gradually available over time, rather than being completely available at the beginning of the learning process.

Your goal for this assignment is to implement an open-ended learning approach for 3D object recognition. We break this assignment down into two parts:

1. The first part is about implementing/optimizing offline 3D object recognition systems, which take an object view as input and produces the category label as output (e.g., *apple*, *mug*, *fork*, etc).
2. The second part of this assignment is dedicated to testing your approach in an open-ended fashion. In this assignment, the number of categories is not pre-defined in advance and the knowledge of agent/robot is increasing over time by interacting with a simulated teacher using three actions: `teach`, `ask`, and `correct` (see Fig. 1).

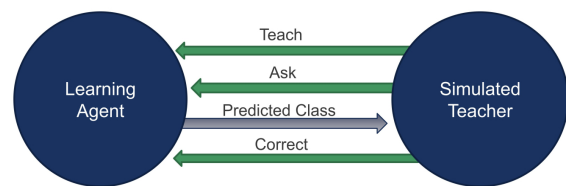


Figure 1: Abstract architecture for interaction between the simulated teacher and the learning agent.

Further details of these assignments are explained in the following sections. To make your life easier, we provide a virtual machine that has all the necessary programs, codes, dataset, libraries, and packages. We also offer template codes for each assignment.

⚠ If you are not familiar with the concept of ROS, please follow the [beginner level of ROS Tutorials](#). For all student, going over all basic beginner level tutorials is strongly recommended.

+ I recommend installing MATLAB on your machine since the output of experiments are automatically visualized in MATLAB. You can download it from [download portal](#) or use [an online version](#) provided by the university.

+ As an alternative, we also provide a python script to visualize the generated MATLAB plots automatically. You can use it as follows: `$ python3 matlab_plots_parser.py -h` to check the instruction

Policies

- Feel free to collaborate on solving the problem but please write your code/report individually. **In particular, do not copy code/text from other students or online resources.**
- **You are not allowed to publish any part of this code online** or claim that you have written it. It does not matter, even if the code is partially used. If you want to publish your results as a scientific paper or use this framework in other projects, contact [Hamidreza Kasaei](#) (hamidreza.kasaei@rug.nl) directly and discuss the case explicitly.

Part I: Offline 3D object recognition setting (50%)

In this assignment, we assume that an object has already been segmented from the scene and we want to recognize its label. We intent to use an instance-based learning (IBL) approach to form new categories. From a general perspective, IBL approaches can be viewed as a combination of an object representation approach, a similarity measure, and a classification rule. Therefore, we represent an object category by storing the representation of objects' views of the category. Furthermore, the choice of the object representation and similarity measure have impacts on the recognition performance as shown in Fig. 2.

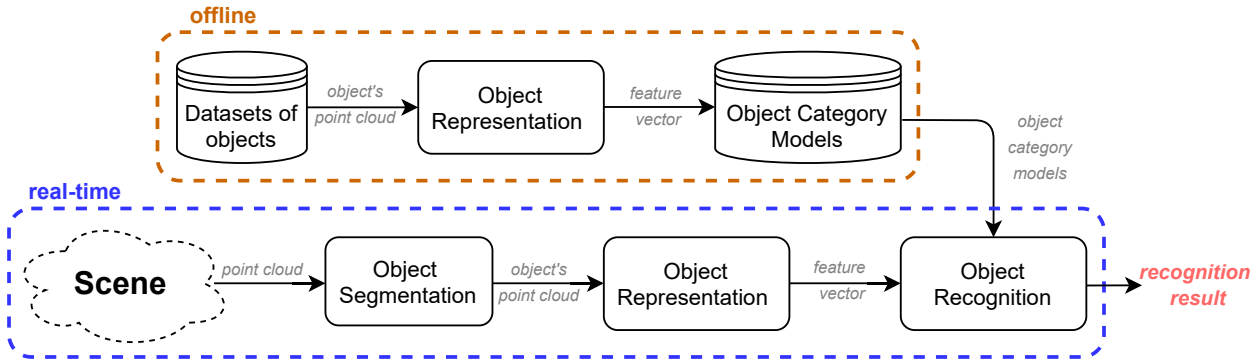


Figure 2: The components used in a 3D object recognition system.

In the case of the similarity measure, since the object representation module represents an object as a normalized histogram, the dissimilarity between two histograms can be computed by different distance functions. In this work, you need to select five out of 14 distance functions that are dissimilar from each other. This policy will increase the chance that different functions lead to different results. The following 14 functions have been implemented and exist in the RACE framework:

Euclidean, Manhattan, χ^2 , Pearson, Neyman, Canberra, KL divergence, symmetric KL divergence, Motyka, Cosine, Dice, Bhattacharyya, Gower, and Sorensen.

✚ For the mathematical equations of these functions, we refer the reader to a comprehensive survey on distance/similarity measures provided by S. Cha (1).

The main intuition behind using instance-based learning in this study is that, IBL serves as a baseline approach for evaluating the object representations used in object recognition. More advance approaches, e.g., SVM-based and Bayesian learning, can be easily adapted.

To examine the performance of your object recognition, we provide a K-fold cross-validation procedure. **K-fold cross-validation** is one of the most widely used methods for estimating the generalization performance of a learning algorithm. In this evaluation protocol, K folds are randomly created by dividing the dataset into K equal-sized subsets, where each subset contains examples from all the categories. In each iteration, a single fold is used for testing, and the remaining nine folds are used as training data. For K-fold cross-validation, we set K to 10, as is generally recommended in the literature. This type of evaluation is useful not only for **parameter tuning** but also for comparing the performance of your method with other approaches described in the literature.

✍ What we offer for this part

- A detail instruction about how to run each of the experiments
- A ROS-based cpp code for 10 fold-cross validation: we have implemented a set of object representation approaches and different distance functions for object recognition purpose. You need to study each approach in depth and optimize its parameters.
- A ROS-based cpp code for 10 fold-cross validation with various deep learning architectures as object representation and a set of distance functions for object recognition purpose. You need to study each approach in depth and optimize its parameters.

- Sample bash scripts for running a bunch of experiments based on GOOD descriptor (hand-crafted), and MobileNetV2 architecture (deep transfer learning), find them out in `rug_kfold_cross_validation/result`).
- A python script to visualize the confusion matrix as the output. Run `python3 matlab_plots_parser.py -p PATH_TO_EXP_DIR/ --offline` to visualize the confusion matrix. You can use `[-h]` to see the instruction.

Your tasks for this part

For this assignment, students will work partly individual and partly in groups of two. Each student needs to optimize one hand-crafted and one deep learning based 3D object recognition algorithm. Therefore, each group will have four set of results. The students will need to write up the report together by discussing the selected approaches and comparing the obtained results in terms of **instance accuracy** ($acc_{micro} = \frac{\# \text{true predictions}}{\# \text{predictions}}$), **average class accuracy** ($acc_{macro} = \frac{1}{K} \sum_{i=1}^K acc_i$), and **computation time**. Note that we need to report average class accuracy to address class imbalance, since instance accuracy is sensitive to class imbalance.

You can think about the following groups:

- (a) **Hand-crafted object representation + IBL approach + K-NN:**
 - list of available descriptors: [GOOD, ESF, VFH, GRSD]
 - distance functions as mentioned above
 - $K \in [1, 3, 5, 7, 9]$
- (b) **Deep transfer learning based object representation + IBL + K-NN**
 - list of available network architectures: [mobileNet, mobileNetV2, vgg16_fc1, vgg16_fc2, vgg19_fc1, vgg19_fc2, xception, resnet50, denseNet121, denseNet169, densenet201, nasnetLarge, nasnetMobile, inception, inceptionResnet]
 - list of available element-wise pooling: [AVG, MAX, APP (append)]
 - distance functions as mentioned above,
 - $K \in [1, 3, 5, 7, 9]$


In this assignment, we use a small-scaled RGB-D dataset to evaluate the performance of different parameter configurations of each approach. In particular, we use **Restaurant RGB-D Object Dataset**, which has a small number of classes with significant intra-class variation. Therefore, it is a suitable dataset for performing extensive sets of experiments to tuning the parameters of each approach.


How to run the experiments

We created a launch file for each of the mentioned object recognition Algorithms. A Launch file provides a convenient way to start up the roscore, and multiple nodes and set the parameters' value (read more about launch file [here](#)).

Before running an experiment, check the following:

- You have to update the value of different parameters of the system in the launch file (e.g., `rug_kfold_cross_validation/launch/kfold_cross_validation.launch`)

 You can also set the value of a parameter when you launch an experiment using the following command: `$ roslaunch package_name launch_file.launch parameter:=value` This option is useful for running a bunch of experiments using a bash/python script

 The system configuration is reported at the beginning of the report file of the experiment. Therefore, you can use it as a way to debug/double-check the system's parameters.

For the hand-crafted based object recognition approaches:

After adjusting all necessary parameters in the launch file, you can run an experiment using the following command:

```
$ roslaunch rug_kfold_cross_validation kfold_cross_validation_hand_crafted_descriptor.launch
```

⊕ For the deep transfer learning based object representation approaches:

After adjusting all necessary parameters in the launch file, you need to open three terminals and use the following commands to run a deep transfer learning based object recognition experiment:

≡ MobileNetV2 Architecture

```
$ roscore
$ rosrug_deep_feature_extraction multi_view_RGBD_object_representation.py mobileNetV2
$ roslaunch rug_kfold_cross_validation kfold_cross_validation_RGBD_deep_learning_descriptor.launch ortho
graphic_image_resolution:=150 base_network:=mobileNetV2 K_for_KNN:=3 name_of_approach:=TEST
```

≡ VGG16 Architecture

```
$ roscore
$ rosrug_deep_feature_extraction multi_view_RGBD_object_representation.py vgg16_fc1
$ roslaunch rug_kfold_cross_validation kfold_cross_validation_RGBD_deep_learning_descriptor.launch ortho
graphic_image_resolution:=150 base_network:=vgg16_fc1 K_for_KNN:=3 name_of_approach:=TEST
```

✍ What are the outputs of each experiment

- Results of an experiment, including a detail summary, and a confusion matrix (see Fig. 3 and 4), will be saved in: `$HOME/student_ws/rug_kfold_cross_validation/result/experiment_1/`

⚠ After each experiment, you need to either rename the **experiment_1** folder or move it to another folder, otherwise its contents will be replaced by the results of a new experiment.

- We also report a summary of a bunch of experiments in a txt file in the following path (see Fig. 5): `rug_kfold_cross_validation/result/results_of_name_of_approach_experiments.txt`

System configuration:

```
-experiment_name = TEST
-name_of_dataset = Restaurant RGB-D Object Dataset
-number_of_category = 10
-orthographic_image_resolution = 150
-name_of_network = /orthographicNet_service
-base_network = vgg16_fc1
-pooling function [MAX, AVG, APP] = MAX
-distance_function = chiSquared
-pdb_loaded = FALSE
-K param for KNN [1, 3, 5, ...] = 1
-running_a_bunch_of_experiments = FALSE
-other parameters = if you need to define more params, please add them here.
```

No.	object_name	ground_truth	prediction	TP	FP	FN	distance
1	Bottle_Object0.pcd	Bottle	Bottle	1	0	0	3.86
2	Bottle_Object1.pcd	Bottle	Bottle	1	0	0	3.492
3	Bowl_Object0.pcd	Bowl	Bowl	1	0	0	3.405
4	Bowl_Object1.pcd	Bowl	Bowl	1	0	0	3.036
5	Bowl_Object2.pcd	Bowl	Bowl	1	0	0	4.057
6	Mug_Object0.pcd	Mug	Mug	1	0	0	2.927
7	Mug_Object1.pcd	Mug	Mug	1	0	0	3.573
8	Mug_Object2.pcd	Mug	Mug	1	0	0	3.097
9	Mug_Object3.pcd	Mug	Teapot	0	1	1	3.416
10	Flask_Object0.pcd	Flask	Flask	1	0	0	6.648

Figure 3: A detailed summary of an experiment: the system configuration is specified at the beginning of the file. A summary of the experiment is subsequently reported. Objects that are incorrectly classified are highlighted by double dash-line, e.g., No. 9.

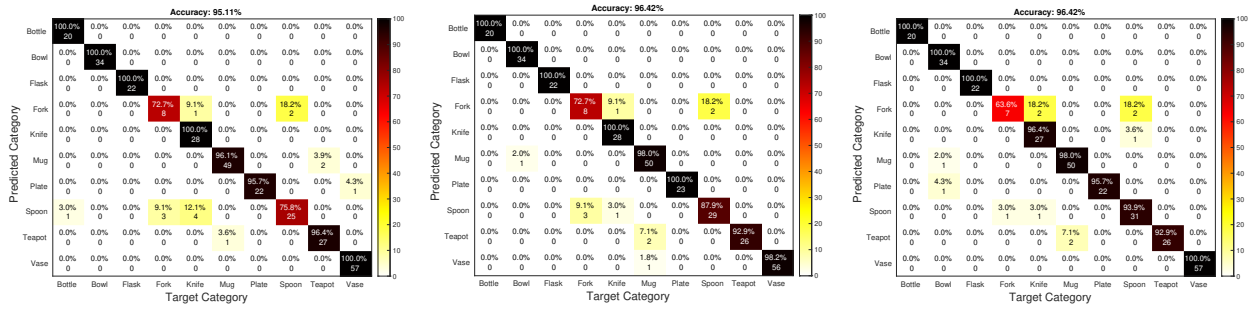


Figure 4: Confusion matrices showing how well each model performed in object recognition task on restaurant object dataset. In each cell of a confusion matrix, we present the percentage and the absolute number of predictions. The darker diagonal cell shows the better prediction by the models.

EXP	Obj.Disc.	#bins	K	Dist.Func.	Ins-Acc	Avg-Class-Acc	Time(s)
1	GOOD	25	1	chiSquared	0.9544	0.9422	2.152
2	GOOD	25	3	chiSquared	0.9381	0.9164	0.9084
3	GOOD	25	5	chiSquared	0.9479	0.9255	1.221
4	GOOD	25	7	chiSquared	0.9381	0.9105	1.273
5	GOOD	25	9	chiSquared	0.9381	0.9105	1.221
6	GOOD	25	1	KLDivergence	0.1303	0.1316	3.035
7	GOOD	25	3	KLDivergence	0.1987	0.2281	1.624
8	GOOD	25	5	KLDivergence	0.1954	0.2246	1.513
9	GOOD	25	7	KLDivergence	0.2117	0.2444	1.619
10	GOOD	25	9	KLDivergence	0.202	0.2273	1.601

Figure 5: A summary of a bunch of experiments for the GOOD descriptor with diffident K and various distance functions: in these experiments, we trained all data first. We then saved the perceptual memory to be used in other experiments.

References

- [1] S.-H. Cha, “Comprehensive survey on distance/similarity measures between probability density functions,” *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 1, no. 4, pp. 300–307, 2007.

Part II: Test your approaches in a systematic open-ended scenario (50%)

The off-line evaluation methodologies are not well suited to evaluate open-ended learning systems, because they do not abide by the simultaneous nature of learning and recognition and also those methodologies imply that the set of categories must be predefined. We, therefore, adopted a teaching protocol designed for experimental evaluation in open-ended learning.

The idea is to emulate the interactions of a recognition system with the surrounding environment over long periods of time in a single context scenario (office, kitchen, etc.). The teacher follows a teaching protocol and interacts with the learning agent using three basic actions:

- **Teach**: used for introducing a new object category to the agent;
- **Ask**: used to ask the agent what is the category of a given object view;
- **Correct**: used for providing corrective feedback in case of misclassification.

Algorithm 1 Teaching protocol for performance evaluation

```

1: Introduce  $Category_1$ 
2:  $n \leftarrow 1$ 
3: repeat
4:    $n \leftarrow n+1$  ▷ Ready for the next category
5:   Introduce  $Category_n$ 
6:    $k \leftarrow 0$ 
7:    $c \leftarrow 1$ 
8:   repeat ▷ question / correction iteration
9:     Present a previously unseen instance of  $Category_c$ 
10:    Ask the category of this instance
11:    If needed, provide correct feedback
12:     $c \leftarrow (c == n) ? 1 : c + 1$ 
13:     $k \leftarrow k + 1$ 
14:     $s \leftarrow$  success in last  $k$  question/correction iterations
15:  until  $((s > \tau \text{ and } k \geq n))$  ▷ accuracy threshold crossed
16:    or (user sees no improvement in success) ▷ breakpoint reached
17: until (user sees no improvement in success) ▷ breakpoint reached

```

Teaching protocol determines which examples are used for training the algorithm, and which are used to test the algorithm (see **Algorithm 1**). The protocol can be followed by a human teacher. However, replacing a human teacher with a simulated one makes it possible to conduct systematic, consistent and reproducible experiments for different approaches. It allows the possibility to perform multiple experiments and explore different experimental conditions in a fraction of time a human would take to carry out the same task. We, therefore, developed a `simulated_teacher` to follow the protocol and autonomously interact with the system. For this purpose, the `simulated_teacher` is connected to a large database of labeled object views. The complete process is summarized in **Algorithm 1** and the overall system architecture is depicted in Fig. 6.

The idea is that the `simulated_teacher` repeatedly picks unseen object views from the currently known categories and presents them to the agent for testing. Inside the learning agent, the object view is recorded in the **Perceptual Memory** if it is marked as a training sample (i.e. whenever the teacher uses `teach` or `correct` instructions), otherwise it is sent to the **Object Recognition** module. The `simulated_teacher` continuously estimates the recognition performance of the agent using a sliding window of size $3n$ iterations, where n is the number of categories that have already been introduced. If k , the number of iterations since the last time a new category was introduced, is less than $3n$, all results are used. In case this performance exceeds a given classification threshold ($\tau = 0.67$, meaning accuracy is at least twice the error rate), the teacher introduces a new object category by presenting three randomly selected objects' views. In this way, the agent begins with zero knowledge and the training instances become gradually available according to the teaching protocol.

⊕ **Breakpoint**: In case the agent can not reach the classification threshold after a certain number of iterations (i.e. 100 iterations), the simulated teacher can infer that the agent is no longer able to learn more categories

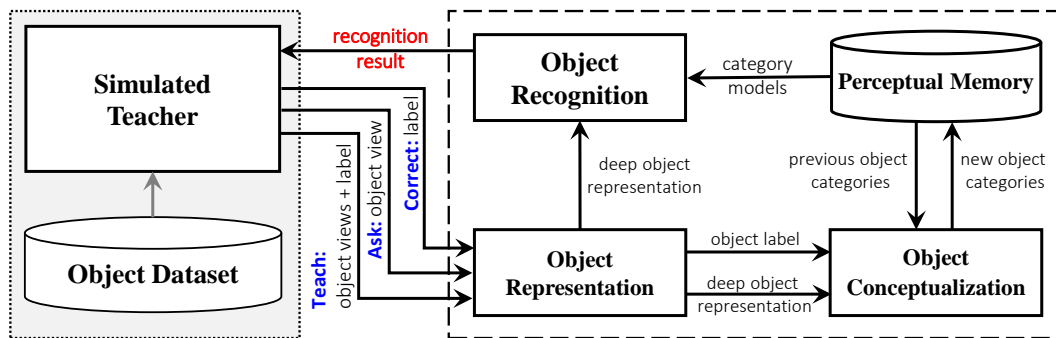


Figure 6: Interaction between the simulated teacher and the learning agent; (left) The simulated teacher is connected to a large object dataset and interacts with the agent by `teach`, `ask` and `correct` actions as shown by blue color; (right) In case of `ask` action, the agent is evaluated by a never-seen-before object. The agent recognizes the object and sends back the result to the simulated user; In the cases of `teach` and `correct` actions, the agent creates a new category model or updates the model of the respective category.

and therefore, terminates the experiment. It is possible that the agent learns all existing categories before reaching the breaking point. In such a case, it is not possible to continue the protocol, and the experiment is halted. In your report, this should be shown by the stopping condition, “lack of data”.

🔗 **Dataset:** In this experiment, one of the largest available 3D object datasets, namely **Washington RGB-D Object Dataset** is used. It consists of 250,000 views of 300 objects and the objects are categorized into 51 categories. Figure 7 shows some example objects from the dataset. We have provided a short version of the dataset that each category only has 200 instances. We have already included the short version of the dataset (3.3 GB) into the virtual machine (\$HOME/datasets/). Both versions are available online: [📁 short-version \(3 GB\)](#) [📁 full-version \(70 GB\)](#)



Figure 7: Sample point clouds of objects in Washington RGB-D Object Dataset.

📁 What we offer for this part

- Provide the simulated teacher code to assess the performance of your code in open-ended settings.
- Provide a set of MATLAB/Python codes to visualize the progress of the agent (related to task #3).
`$ python3 matlab_plots_parser.py -p PATH_TO_EXP_DIR/ --online`
- A bash script for running a bunch of experiments (find it out in `rug_simulated_user/result` folder).

📁 Your tasks for this part

- Based on the obtained results in the previous part (10-fold cross-validation experiments), select **the best system configuration for both hand-crafted and deep transfer learning approaches** (i.e., object representation + distance function). For each of the selected approaches, update the parameters of the simulated teacher in the launch files accordingly.
- Since the order of introducing categories may have an effect on the performance of the system, you have to perform 10 experiments and report all 10 experiments plus the avg+std in a table (10 experiments for hand-crafted and 10 for deep-learning).
- Visualize the following plots for **the best learning progress of hand-crafted and deep transfer learning approaches** (as an example, see Fig. 8), and compare them together (for further details on evaluation metrics and plots, please check out the OrthographciNet paper, which is available in the Nestor):
 - protocol accuracy vs. #question/correction iterations (explain first 200 iterations)
 - number of learned category vs. #question/correction iterations
 - global classification accuracy vs. #question/correction iterations
 - number of stored instances per category

➕ It should be noted that, instead of having diffident plots for each approach, you can visualize all your results together using the provided python script. Such visualization is really useful to analyse and compare the approaches. **More information about the python parser is available on Nestor under “Practical assignments” tab.**

- The `protocol_threshold` parameter, τ , defines how good the agent should learn categories. For example, $\tau = 0.67$ means the recognition accuracy is at least twice better than the error. Therefore, it can influence on all evaluation metrics. For each of the selected approaches, you need to perform **only three experiments** by setting $\tau \in [0.7, 0.8, 0.9]$, e.g., `protocol_threshold:=0.7`, and `random_sequence_generator:=false` to have fair comparison. Finally, you need to analyse the effect of τ based on the obtained results.

📁 How to run the experiments

Similar to the offline evaluation, we created a launch file for hand-crafted and deep transfer learning based algorithms. However, before running an experiment, check the following items:

- You have to **update the value of different parameters** of the system in the relative **launch** file.

+ The system configuration is reported at the beginning of the report file of the experiment. Therefore, you can use it as a way to debug/double-check the system's parameters.

☉ For hand-crafted based object representation approaches:

After setting a proper value for each of the system's parameter, you can run an open-ended object recognition experiment using the following command:

```
$ roslaunch rug_simulated_user simulated_user_hand_crafted_descriptor.launch
```

☉ For deep learning based object representation approaches:

Similar to the offline evaluation for deep learning based approaches, you need to open three terminals and use the following commands to run an open-ended object recognition experiment for an specific network architecture:

≡ MobileNetV2 Architecture

```
$ roscore
$ rosrn rug_deep_feature_extraction multi_view_RGBD_object_representation.py mobileNetV2
$ roslaunch rug_simulated_user simulated_user_RGBD_deep_learning_descriptor.launch ortho
  graphic_image_resolution:=150 base_network:=mobileNetV2 K_for_KNN:=7 name_of_approach:=TEST
```

≡ VGG16 Architecture

```
$ roscore
$ rosrn rug_deep_feature_extraction multi_view_object_RGBD_representation.py vgg16_fc1
$ roslaunch rug_simulated_user simulated_user_RGBD_deep_learning_descriptor.launch ortho
  graphic_image_resolution:=150 base_network:=vgg16_fc1 K_for_KNN:=7 name_of_approach:=TEST
```

⚠ To have a fair comparison, the order of introducing categories should be same in both approaches. Therefore, we design a Boolean parameter named `random_sequence_generator` that can be used for this purpose. Check out the script we have provided for more details.

✍ What are the outputs of each experiment

- Results of an experiment, including a detail summary and a set of MATLAB files (see Fig. 8), will be saved in: `$HOME/student_ws/rug_simulated_user/result/experiment_1/`

⚠ After each experiment, you need to either rename the `experiment_1` folder or move it to another folder, otherwise its contents will be replaced by the results of a new experiment.

- The system also reports a summary of a bunch of experiments as a txt file in the following path : `rug_simulated_user/result/results_of_name_of_approach_experiments.txt`

+ Each time you run an experiment, the experiment results will be automatically appended to the log file. After running a bunch of 10 experiments, you have to report the content of the log file as a table in your report, compare the obtained results, and visualize the output of the best experiment for hand-crafted and deep transfer learning experiments (as an example see Fig. 8).

✍ Extra credit

To be eligible for the extra credit points, your approach (object representation or recognition) should be different than the provided sample codes. We will evaluate your object recognition approach using the same simulated teacher code. We will add 0.5 to the final score of the student who achieves the highest performance, 0.35 points to the student who achieves the second place, and 0.20 points to the student who achieves third place. We will compute the performance of your algorithm ourselves (code that does not run will be disqualified from the contest). This reward is designed to encourage you to experiment with different algorithms and hyperparameter settings to obtain the best performance.

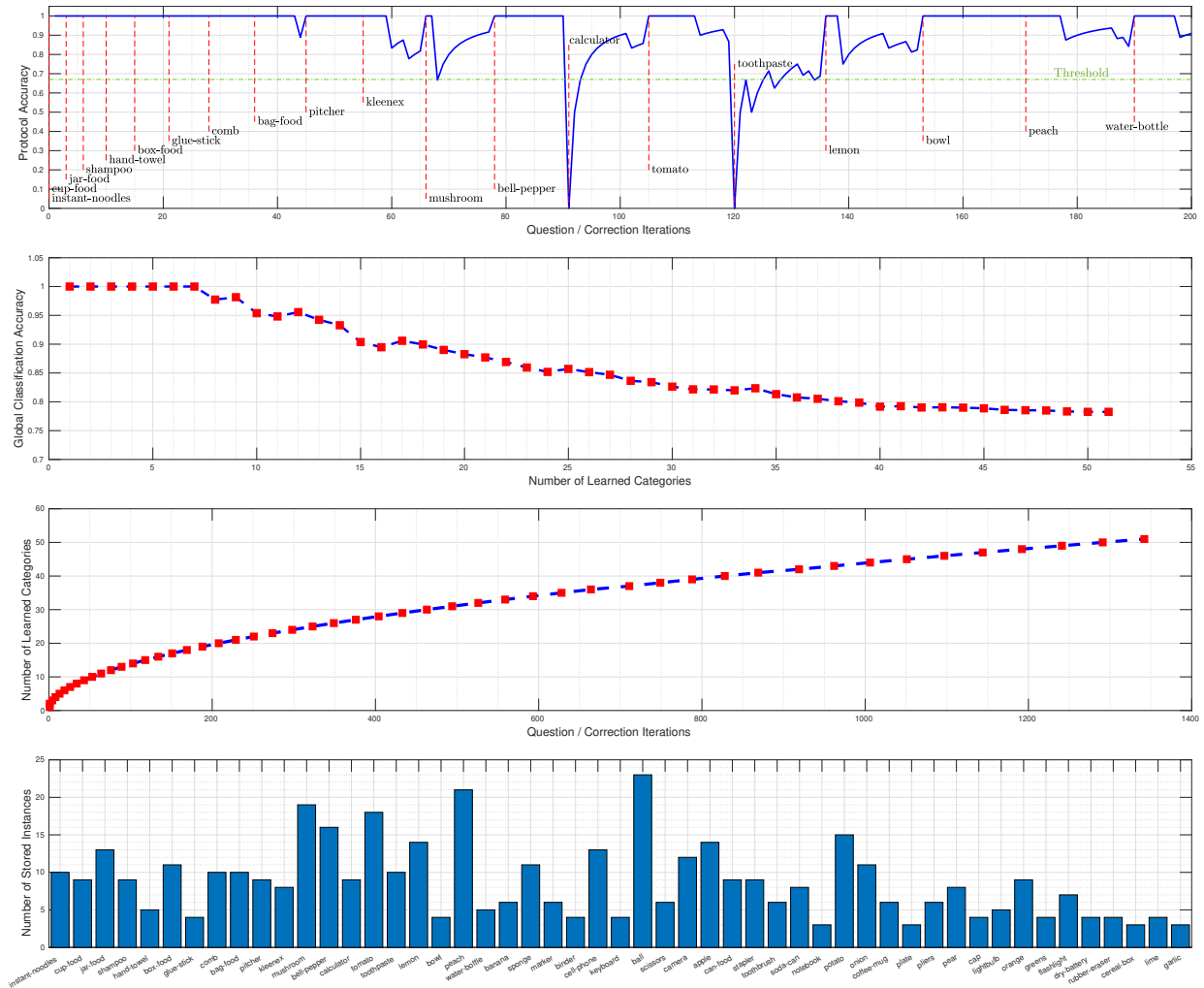


Figure 8: Summary of an example open-ended experiment: (*top*) Evolution of protocol accuracy over first 200 iterations. In this experiment, the protocol threshold is set to 0.67, as shown by the green horizontal dashed line. Once the agent has classified all learned categories at least once, and the protocol accuracy is higher than the threshold, a new category is introduced, which is shown by a red dashed line and a category name. (*second*) This plot shows global classification accuracy as a function of number of learned categories for the same experiment. It can be seen that the agent was able to stay above the protocol threshold during the entire experiment, indicating the agent can learn many more categories. (*third*) This graph represents how fast does the agent learn by representing the number of learned categories as a function of the number of question/correction iterations. (*bottom*) This graph shows the number of instances stored in each category, i.e., the three instances provided at the introduction of the category together with the instances that had to be corrected somewhere along the experiment run. The ball category apparently was the most difficult one, requiring the largest number of instances.

Submission

A report (i.e., up to four pages – two pages for the first part and two pages for the second part – [IEEE conference format](#), containing all figures, tables, and references) has to be delivered. You need to include an “Authors’ Contributions” section at the end of the report explaining how did you divide the work among the members, and what are the contributions of each author. We expect all authors contribute equally to the assignment. Submit your assignment in as a pdf file named `group_number_prj1.pdf`



Do not delete your results after submitting the report. We may ask you to send us your `rug_kfold_cross_validation` and `rug_simulated_user` packages and the obtained results.