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

Introduce $Category_1$
$n \leftarrow 1$
repeat
  $n \leftarrow n+1$ ▷ Ready for the next category
  Introduce $Category_n$
  $k \leftarrow 0$
  $c \leftarrow 1$
  repeat ▷ question / correction iteration
    Present a previously unseen instance of $Category_c$
    Ask the category of this instance
    If needed, provide **correct** feedback
    c $\leftarrow (c == n)$? 1 : $c+1$
    $k \leftarrow k+1$
    $s \leftarrow$ success in last k question/correction iterations
  until (($s > \tau$ **and** $k >= n$) ▷ accuracy threshold crossed
    or (user sees no improvement in success)) ▷ breakpoint reached
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. 1.

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

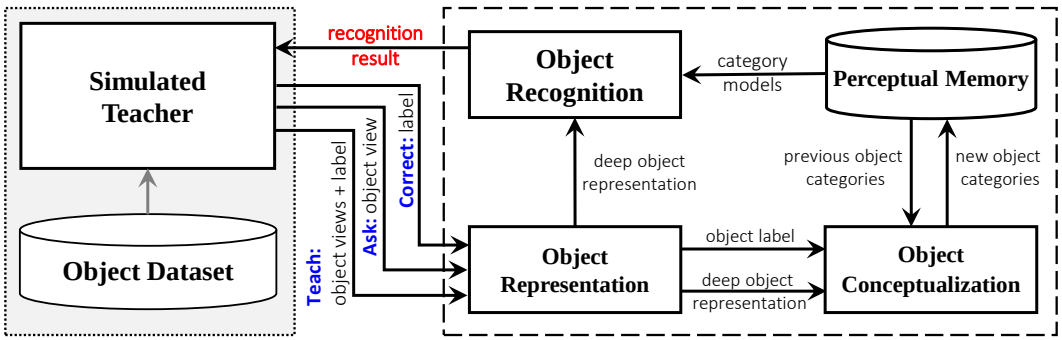

Figure 1: Interaction between the simulated teacher and the learning agent; (*left*) The simulated teacher is connected to a large object dataset and interacts with the agent by `teach`, `ask` and `correct` actions as shown by blue color; (*right*) In case of `ask` action, the agent is evaluated by a 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 2 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 2: 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. 3), 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
$ rosrun rug_deep_feature_extraction multi_view_object_representation.py mobileNetV2
$ roslaunch rug_simulated_user simulated_user_deep_learning_descriptor.launch
orthographic_image_resolution:=150 base_network:=mobileNetV2 name_of_approach:=TEST
```

VGG16 Architecture

```
$ roscore
$ rosrun rug_deep_feature_extraction multi_view_object_representation.py vgg16_fc1
$ roslaunch rug_simulated_user simulated_user_deep_learning_descriptor.launch
orthographic_image_resolution:=150 base_network:=vgg16_fc1 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. 3), 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. 3).

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.

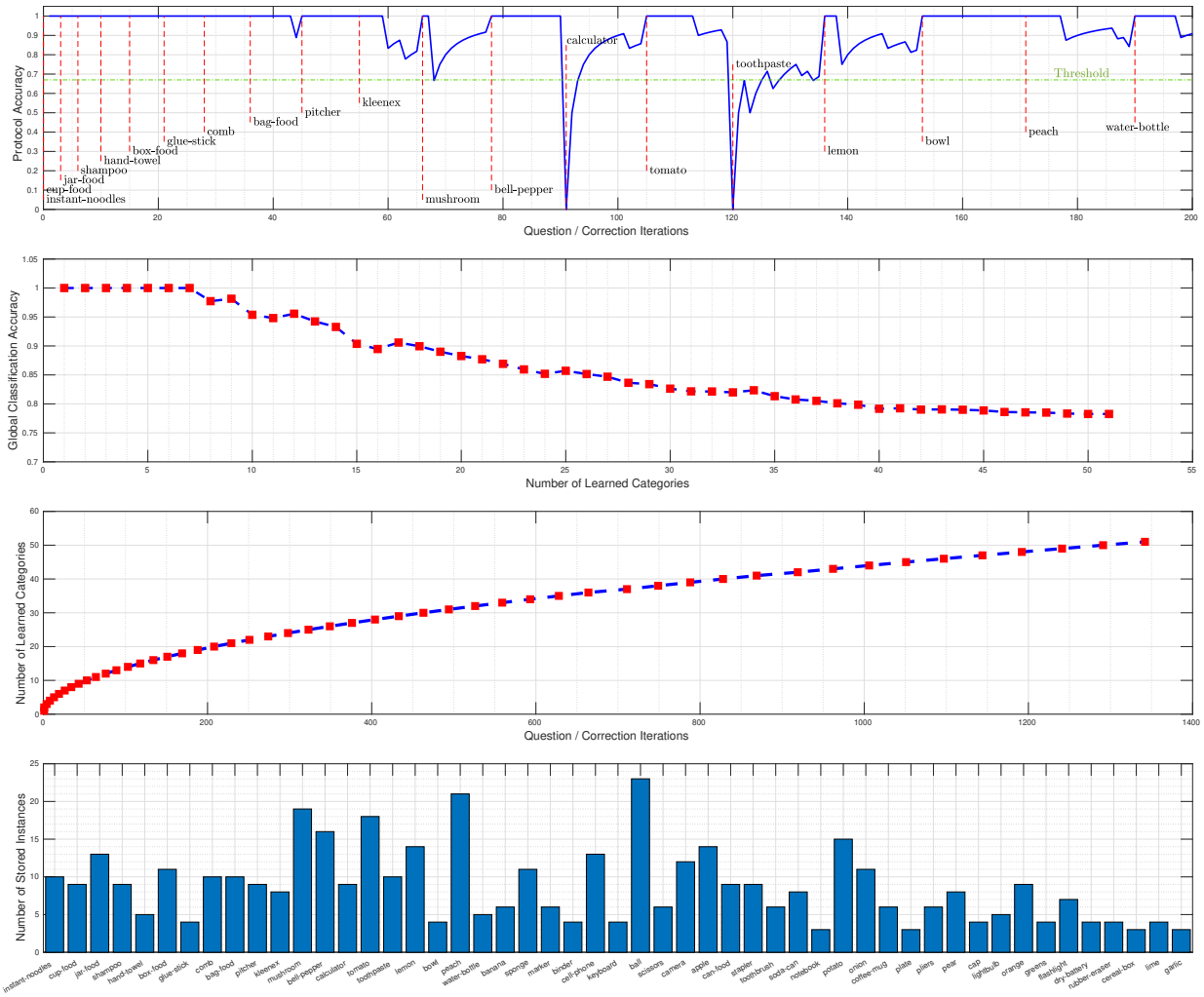

Figure 3: Summary of an example open-ended experiment: (*top*) Evolution of protocol accuracy over first 200 iterations. In this experiment, the protocol threshold is set to 0.67, as shown by the green horizontal dashed line. Once the agent has classified all learned categories at least once, and the 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.