Milestone 3: Object Recognition and Localisation

(Last modified: 12 Sep 2022)

In this milestone (Week 6 and 7), you will program your robot to perform object recognition and localisation. Various fruits will be placed in the arena and your robot has to detect the fruits from the camera images. For this, you will implement a deep learning solution by collecting a dataset, training a deep neural network, and testing your trained detector. Then, using the results of the detection, you will need to estimate the poses (x-y location) of the fruits in the arena, which will then be validated against the ground truth poses.

Setting Up

Client (Your laptops)

Duplicate your M2 folder and rename it as M3. Delete operate.py in M3. Download the zip folder for M3 in Moodle and unzip them into the new folder. Edit the new operate.py file by importing your teleoperating codes (& any other part you may have modified).

Launch a terminal and install a few packages:

```
pip install h5py tqdm torch torchvision
pip install -U PyYAML
```

Verify the installation by running `python`, and then `import torch`
If there is no error, exit python by typing Ctrl+Z or exit().

There are a few large files to be generated in this milestone. As GitHub has an upload limit, you may want to create a `.gitignore` file (can be edited with notepad) to avoid uploading certain files to GitHub.

Activities

Part 1: Data Collection

1. With your robot, take plenty of pictures of the fruits (redapple, greenapple, orange, mango, capsicum) and the arena separately. For the fruits, use a simple background like the picture below. Be sure to take pictures of multiple orientations of the fruits and multiple angles for the arena.

2. Write a script to generate a sizable dataset using the pictures that you have taken. A suggested workflow is as follows:
    - Remove the background of the fruit pictures by setting the background pixel to 0. This is easier if your fruit pictures have a uniform background.
    - Superimpose the fruit image onto the arena pictures. You can randomise the placement of the fruit within the image, as well as the size of the fruit, in order to introduce variability in terms of the location and depth of the fruits. With multiple pictures of arena and fruit orientations, together with random fruit scales and placements, you should be able to easily generate thousands of images.
    - For every image, a label is required (make use of the cropped image). The label has the same width and height as the image (but only 1 channel), with all pixels as 0 (background) except the pixel locations of the fruits. Assign the value of the pixels as follows:
      background: 0, redapple: 1, greenapple: 2, orange: 3, mango: 4, capsicum: 5

3. Save your dataset into “network/scripts” according to the folder hierarchy as follows:

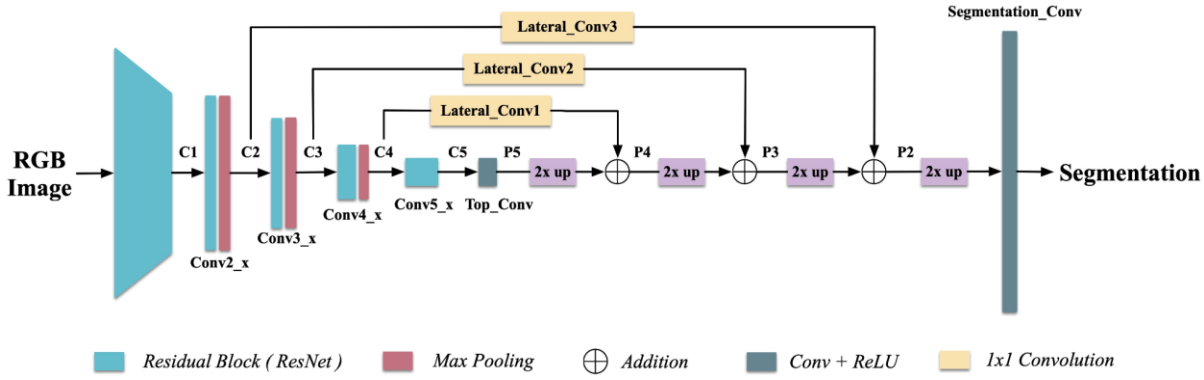
```
network/scripts/dataset/images/image_0.png
network/scripts/dataset/images/image_1.png
…
network/scripts/dataset/labels/image_0_label.png
network/scripts/dataset/labels/image_1_label.png
…
```

Tips:

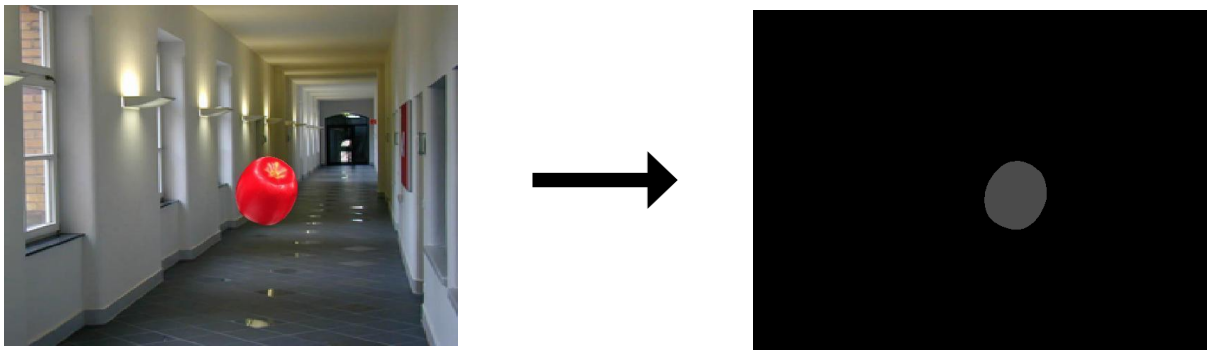
- You can use MATLAB or any other language for data generation. (Note that the model training for next activity is still be in python)
- For removing background, you can use any tools you like, such as an online tool, or write your own script.
- An image can contain more than 1 fruit.
- You can also obtain fruit pictures from the Internet, as well as other random background pictures (instead of the arena) to increase the size of your dataset and improve your trained detector’s robustness.

Part 2: Train your fruit detector neural network model

You are provided with ResNet-18 as the model, which is a deep convolutional neural network proposed for ImageNet classification.. The structure of the ResNet18 model is specified in `res18_skip.py`.



The code provided to you will train the network to accept an input image and generate label prediction for every pixel of the image. This is known as image segmentation.



1. After you have generated your data, run
   ```
   cd network/scripts
   python split_dataset.py --training_ratio 0.8
   ```
   This will separate the dataset randomly into a training set containing 80% of all images and an evaluation set containing 20% of all images, specified by "train.hdf5" and "eval.hdf5" generated under "dataset" folder, which will be used to train your neural network.

2. To train the neural network with the data you generated, run the commands below:
   ```
   cd network/scripts
   python main.py --dataset_dir dataset --model_dir model
   ```

3. You can change the default parameters by adding flags, such as `--lr 0.05`, when running main.py. Parameters that you can change, what they represent and their default values are specified in `args.py`.

4. You will see some information printed regarding how a model performs for training and evaluation during each epoch. Once training is done, a best performing model will be saved as "/network/scripts/model/model.best.pth". This model should be able to detect the different targets (fruits) and segment them from the background. If you want to train new models, delete old checkpoints (model.pth) before training.

5. If you would like to train your neural network on Google Colab, upload the ECE4078_2022_Lab_M3_Colab folder with your dataset included (the hdf5 files) to your Google Drive. Open main_colab.ipynb in Colab and then run its cells (you can change the default parameters in args.py). If you would like to use GPU with Colab, in the top menu, go to "Runtime" -> "Change runtime type" -> "Hardware accelerator" -> select "GPU" in the dropdown menu. After the training is done, you can download the generated best model "model.best.pth" to “network/scripts/model/”.

6. To test the performance of your model, run operate.py, position the fruit(s) in the field of view of the robot, and then press “p” to run the detector. You may need to change the path to the best performing model by running operate.py with the --ckpt flag. Your network should segment the input image into different classes (background, redapple, greenapple, orange, mango and capsicum).

Tips:

- The materials provided to you are only a guideline to assist you to train a basic target detector that can provide sufficient performance for you to complete this milestone. Feel free to use your own codes and models, such as exploring other state-of-the-art object detection models like YOLOv5.
- If you decide to use other models, you can make your own file hierarchy structure but please make sure you follow the same naming convention and weight file (.pth) for evaluation purposes.

Part 3: Estimating Target Pose

You are required to estimate the targets' pose, using your detector's results and your robot's current pose. For example, if your robot is at the origin and there is an apple in the field of view of the robot's camera, you have to estimate the pose of the apple (its x-y location) based on your robot's current pose (x=0, y=0, theta=0) and your detector's segmentation prediction. Then, your target pose estimation will be evaluated against the ground truth.

To estimate pose of targets, you will need to run the SLAM component before running the target detector, so that the robot's pose is known. Every time you want to perform object detection, press "p" to run the detector, then press "n" to save the robot's current pose estimated by SLAM as well as the corresponding detector's segmentation labels These segmentation labels appear all black, similar to the images in the "labels" folder when generating the data. You can press "i" to save the raw image in addition for visual debugging of the pose estimation. The segmentation labels are saved as "lab_output/pred_x.png" and the corresponding robot's pose is saved in "lab_output/images.txt".

Complete TargetPoseEst.py to estimate the locations of the redapple, greenapple, orange, mango and capsicum based on the detector's outputs and the robot's poses. Replace Lines 87-93 with your own codes to compute the target pose using the robot's pose and the detector's output. If you are using other kinds of models such as YOLO, feel free to edit the codes outside the TODO section to merge with your model.

While performing object detection, you may have multiple images containing the same fruit. For example, if you have N images (different robot pose) that contain orange in the field of view, you would have generated N orange pose estimations. However, there is only 1 of each fruit type in the marking map, so you can only output at most 1 estimation per target type in the estimation output. Replace Lines 117-152 with your own codes to merge the estimations with a better way than simply taking the first estimation. The TargetPoseEst.py generates an estimation result file as "lab_output/targets.txt", which is in the same format as the groundtruth maps.

You can use CV_eval.py to evaluate performance of your target pose estimation. Run

```
python CV_eval.py TRUEMAP.txt lab_output/targets.txt
```

This computes the Euclidean distance between each target and its estimation (the estimation error) and returns the average estimation error over all targets. If more than 1 estimation are given for a target type, the first estimation will be used in the evaluation.

Demo & Marking

Demo for M3 is on Week 8. The demo procedure is as follows:

1. Ensure that the folder “pibot_dataset” is empty. Run operate.py.
2. Your demonstrator will give you a certain robot pose (x,y,theta).
3. Position your robot in the arena according to the given pose.
4. Press “i” to take an image. The camera view should contain at least 1 fruit.
5. Repeat step 2-4 for other new robot poses given by your demonstrator.
6. When all robot poses are done, exit the program.
7. In your M3 folder, ensure that TargetPoseEst.py is in the folder, and your best model is saved as “network/scripts/model/model.best.pth”.
8. Temporarily move your “dataset” folder out of your M3 folder. Then, zip the whole M3 folder and submit to Moodle > Assessments > Lab M3 Malaysia.
9. If your zip folder fails to be uploaded due to Moodle size limit, upload it to your Google Drive. Then, upload a text file to Moodle containing the Drive link.

For marking, we will first load your target detector using your model.best.pth. Then, we will pass the images that you have taken into the detector to generate the predictions. Using these predictions, as well as the known robot poses, we will run your TargetPoseEst.py to generate the fruit pose estimations, which will be compared against the ground truth.

Note that for simplicity of M3, we allow the use of known (ground truth) robot poses associated with each image so that your estimation performance will only depend on your neural network's performance and your pose estimation script. However, in the final demo in Week 12 and Week 13, you will be using the robot poses that are estimated by your SLAM.

Demo Procedure (Updated)

- The arena will have 10 Aruco markers and 5 fruits (1 redapple, 1 greenapple, 1 orange, 1 mango, 1 capsicum).
- Demonstrators will not specify robot pose for demo. You can choose the robot poses that you prefer to take the pictures.
- You must take at least 5 pictures, at most 25 pictures.
    a. All pictures must have at least 1 fruit.
    b. All pictures must be taken from a unique robot pose (i.e. no repetition).
    c. At least 1 picture must have 2 or more different types of fruits.
    d. At least 1 picture must have the target being 0.8m or further away from the robot.

- You need to specify the ground truth of the chosen poses in `lab_output/images.txt`.
- You can discuss/prepare/test beforehand, but for submission, you must take the pictures live in front of demonstrators (cannot use old pictures), and submit the whole M3 folder right after the demo.
- Submitted model.best.pth will be checked for plagiarism. Hardcoded fruit pose will result in 0 marks.
- Your score will be calculated as follows:

```
Score = (1-AvgEstimationError)/(1-0.025) x 80 + NumberOfFruitTypeFound x 4
```