CS 4610/5335: Robotic Science and Systems (Spring 2024) X. Zhi Tan
Northeastern University version: 1.0.0

HW 4: Machine Learning & Grasp Detection

Please remember the following policies:

- Submissions should be made electronically via the Canvas. Please ensure that your solutions for both the written and/or programming parts are present and zipped into a single file.
- Solutions may be handwritten or typeset. For the former, please ensure handwriting is legible.
- You are welcome to discuss the programming questions with other students in the class. However, you must understand and write all code yourself. Also, you must list all students (if any) with whom you discussed your solutions to the programming questions.
- Any use of generative AI must follow the class' generative AI policy.

All of the code for this assignment is in Python, and located within 'hw4_code_YYYYMMDD.zip'. We will create a new conda environment for this assignment. Recommended steps:

- Download `hw4.zip` from Canvas. Unzip the file to your preferred directory (e.g., `/Users/Zhi/RSS/HW4`).
- Create new conda environment - `create -n rss-hw4 python=3.9`
- Activate conda environment - `conda activate rss-hw4`
- Navigate to the directory and install the required packages - `python -m pip install -r requirements.txt`

Please add comments to your code to help the graders understand what you are doing. If you make a mistake, you are much more likely to receive partial credit if the code is understandable.

Q1. Warm-up Questions (**2 points**).
Before we dive into grasp detection, let's start by training a simple model to classify clothing in the Fashion-MNIST dataset. The dataset consist of 28x28 grayscale images that you will classify into ten groups. First navigate to the `warmup` directory and run `python trainer.py`. This will train a single layer fully connected layer for 5 epochs and should have a final accuracy of less than 0.85 on the test set. In this warmup question, you will modify the `model.py` file to improve the model's performance. *We choose 5 epochs to decrease training time and in real applications you would train until certain criteria is met.*

(a) Modify the `model.py` by adding convolution layers and achieve an accuracy of more than 0.85 on the test set. While we include a recommended architecture below, any convolutional architecture that achieves an accuracy above 0.85 is acceptable.

Suggested Architecture: Conv2d(32, kernel=3, stride=1, padding=1) → ReLU → Pool(kernel=2, stride=2) → Conv2d(64, kernel=3, stride=1, padding=1) → ReLU → Pool(kernel=2, stride=2) → Flatten → FC → Softmax

For an introduction to creating neural networks with PyTorch, see this guide. If you want to look up specific pytorch modules, try searching the docs

Q2. Grasp Simulator (**2 points**).
In the `grasping` directory, run the command `$python simulator.py` to view the grasp simulator via the Pybullet graphical user interface. You should see a window pop up that looks like Fig. 1. The simulator contains a Panda Arm and a square workspace where small objects are randomly placed. After every reset, the robot arm attempts a grasp with the gripper aligned along x- or y-axis. An RGB camera (not visible) is positioned to view the workspace from above, and the camera feed is shown in the top left panel. You can zoom in and out with the mouse scroll; hold down the control key and right mouse button to rotate your view of the scene.

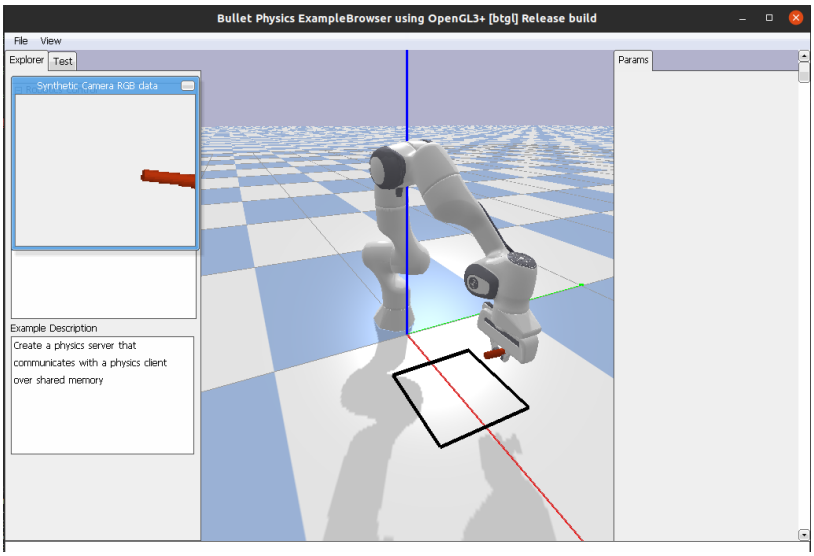

Figure 1: Grasp simulator in Pybullet.

(a) Describe the benefits of using a simulator to generate datasets of grasp attempts. Consider the two factors of speed and safety.

(b) Do you notice any unrealistic physics that occurs on some grasps? Hint: rotate your view to be parallel with the ground plane.

The simulator from above was used to collect a dataset of 1,000 **successful** grasps, split into a train and validation set. For each grasp, a 64×64 pixel RGB image is captured of the scene and we record the pixel location (px, py) and rotation (0 or 90 degrees) of the gripper. In the following sections, you will build and train a network to predict successful grasps given a top-down RGB image of the scene. We have designed this assignment so that the network can be trained and evaluated entirely on CPU.

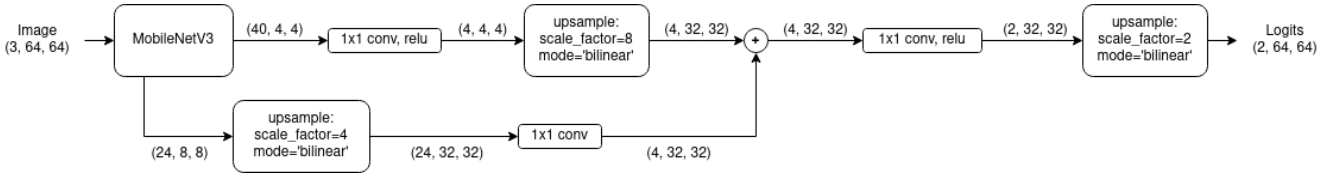

Figure 2: Unet with MobileNetV3 backbone. Architecture inspired by Fig. 10 of "Searching for MobileNetV3" (`https://arxiv.org/pdf/1905.02244.pdf`). Intermediate tensor shapes are labeled without batch dimension.

Q3. Create Grasp Prediction Network (**3 points**).
You will now create a convolutional network to predict successful grasps given a top-down RGB image of the scene. The network will output an image with the same size as the input image, where each pixel has two channels corresponding to the two different gripper rotations (0 degrees and 90 degrees about the z-axis). In other words, the network outputs a distribution over pixel location and gripper rotation for a **successful** grasp. At inference time, the argmax over pixels and channels can be used to determine the best gripper location and rotation. The network is trained to minimize a cross entropy loss over the output distribution.

(a) Complete the implementation of `MobileUNet.__init__` and `MobileUNet.forward` in `trainer.py` using the model diagram in Fig. 2. This is a fully convolutional network with a MobileNet backbone, which is designed to be run efficiently on the CPU. The code to calculate the cross-entropy and predict a successful grasp location is already provided for you, but you may wish to review the implementations at lines 141 and 73 respectively.

(b) The network is designed with two pathways emerging from the MobileNet backbone. The pathways are intended to carry information with different spatial resolutions. What local information is important for predicting grasp success? Is there any global information that is needed? If not, can you think of a task where global information would be needed?

Q4. Training and Evaluating Network (**2 points**).

(a) Run the training script with the command `$python trainer.py`. It should take around 5 minutes to train for 30 epochs on your CPU (if you have a GPU, it will automatically use it). While it is training, you can monitor the loss curves by viewing the image file 'loss_curves.png' and see predictions by viewing 'predictions.png'. It is expected that the validation loss curve will be very noisy (**+1 Extra Credit point if you can explain why!**) so only the best performing weights are saved. The data in Figures 3,4 are approximately what you should see (achieving a validation loss of around 3), but the code is not perfectly repeatable.

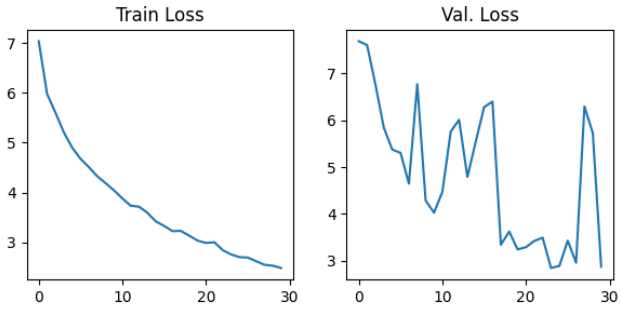

Figure 3: Example loss curves.

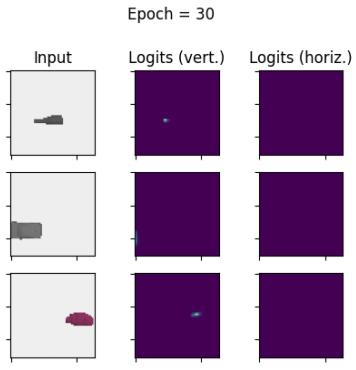

Figure 4: Example predictions.

(b) Explain why the validation loss is not a perfect indicator of how well the network can predict grasps. Hint: could the network predict a valid grasp location that results in a large loss term (e.g. false negative)?

(c) To get a better sense of the model performance, we can evaluate it directly in the simulator where the data was generated. Run the command `$python evaluate_model.py` to see the model perform 50

grasp attempts. What success rate is achieved (report the final estimate which is averaged over all 50 attempts)? Do you notice anything different about the object placements here (compared to what you see in 'predictions.png')?

Q5. Adding Data Augmentation (**3 points**).
You may have noticed a discrepancy between the object placement in the dataset and in the evaluation simulator. In the dataset, the objects are axis-aligned, while the evaluation simulator places objects at arbitrary rotations. To improve evaluation performance, you will add apply random rotations during training. However, you must be careful because the image and label need to be transformed jointly.

(a) Implement the method `GraspDataset.transform_grasp` in 'dataset.py'. For more details, see the method docstring.

(b) Re-train your network with the rotation data augmentations. Run the evaluation script again (it will automatically use the new network weights). Report the success rate.

(c) Say we also wanted to improve generalization to translations of the objects in the scene. Describe in writing what transformations you would apply to the image, gripper rotation, and pixel location to augment the data in this way.