CS 4610/5335: Robotic Science and Systems (Spring 2023) Northeastern University Robert Platt

HW 6: 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 (but *not* the written 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.

All of the code for this assignment is in Python, and located within 'hw6.zip'. A guide for how to install necessary libraries is found in 'README.md'. **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. Grasp Simulator (**2 points**).

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.

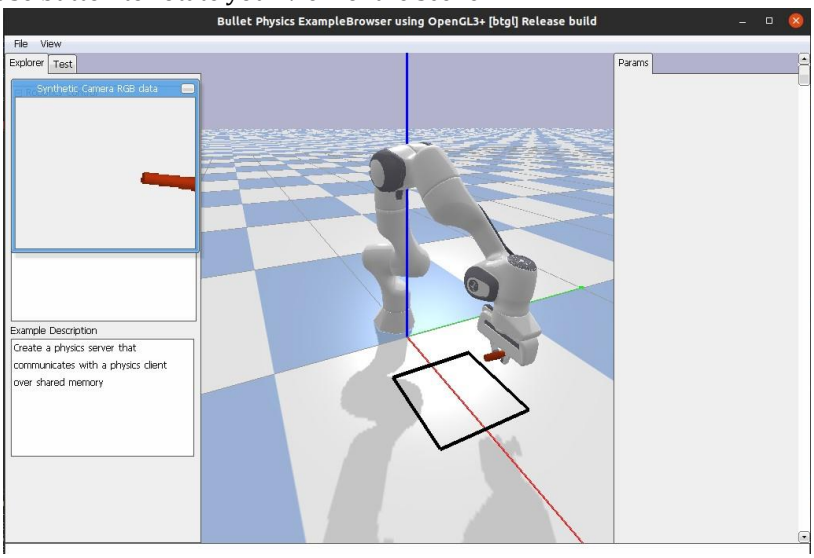


Figure 1: Grasp simulator in Pybullet.

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

i. Speed:

1. Faster data collection: Producing grasp attempts in a simulator can be done far more quickly than performing trials in real life. Physical configurations frequently call for labor-intensive manual resets, upkeep, and troubleshooting. Simulators, on the other hand, can perform several grip attempts quickly one after the other and instantaneously reset environments.
2. Parallelization: Simulators have the ability to run numerous instances concurrently on various processing units, greatly accelerating data collection. Contrarily, using physical robots for parallel studies necessitates multiple setups, which raises the expense and complexity.
3. Parameter exploration: Simulators make it simple to change the environment and robot parameters, allowing researchers to quickly evaluate different setups. Understanding the impact of various parameters on the grip success rate is aided by this flexibility.

ii. Safety:

1. Risk-free testing: Simulators offer a secure environment for testing new setups or algorithms without running the risk of endangering the robot or nearby objects. Physical grab efforts that are not well-tuned or thoroughly tested by the algorithm may result in mechanical failures or accidents.
2. Human safety: In simulators, there's no risk to human operators or bystanders during the experiments. Real-world experiments with robotic arms can pose safety hazards if the robot malfunctions or the environment is not adequately controlled.
3. Reduced wear and tear: Repeatedly attempting to grip objects with physical robots cause wear and tear issues on the hardware. Simulators prevent this by enabling testing of algorithms without affecting the robot's life span.

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

- Yes. Especially when zooming in.
- Observations:
  - It sometimes slows down and something weird happens
  - When arm presses on the top of object, the object seems to go a bit below the ground
  - Sometimes the object sticks to the gripper even though the gripper is not holding on firmly on the object

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.

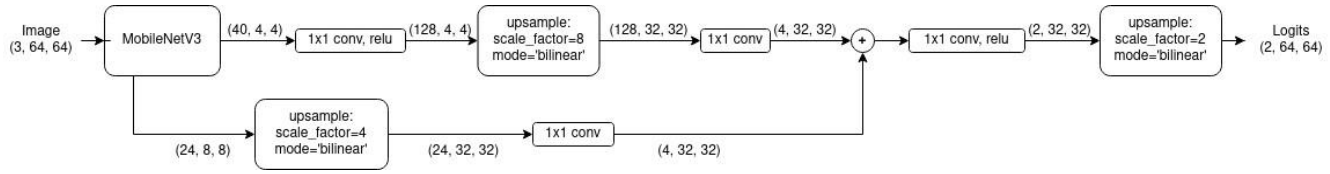


Figure 2: Unet with MobileNetV3 backbone. Architecture inspired by Fig. 10 of "Searching for MobileNetV3" (https://arxiv.org/pdf/1905.02244.pdf). Intermediate tensor shapes are labeled without batch dimension.

Q2. 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 wishS to review the implementations at lines 141 and 73 respectively.

```
#########################
# define other layers here
#########################
self.up_1 = nn.Sequential(
    nn.Conv2d(40, 128, kernel_size=1, stride=1, padding=0),
    nn.ReLU()
)
self.up_2 = nn.Sequential(
    nn.Conv2d(128, 4, kernel_size=1, stride=1, padding=0)
)
self.up_3 = nn.Sequential(
    nn.Conv2d(4, 2, kernel_size=1, stride=1, padding=0),
    nn.ReLU()
)

self.up_4 = nn.Sequential(
    nn.Conv2d(24, 4, kernel_size=1, stride=1, padding=0)
)
```

S

```
########################################
# implement additional steps in forward pass
########################################s
# Upsample and convolutions
x_up_1 = self.up_1(x_narrow)  # tensor of shape (B, 128, 4, 4)
x_up_1 = F.interpolate(x_up_1, scale_factor=8, mode='bilinear',
                       align_corners=True)  # tensor of shape (B, 128, 32, 32)

x_up_2 = self.up_2(x_up_1)  # tensor of shape (B, 4, 32, 32)

x_up_3 = self.up_3(x_up_2)  # tensor of shape (B, 2, 32, 32)
x_up_3 = F.interpolate(x_up_3, scale_factor=2, mode='bilinear',
                       align_corners=True)  # tensor of shape (B, 2, 64, 64)

x_up_4 = F.interpolate(x_wide, scale_factor=4, mode='bilinear',
                       align_corners=True)  # tensor of shape (B, 24, 32, 32)
x_up_4 = self.up_4(x_up_4)  # tensor of shape (B, 4, 32, 32)

x_combined = x_up_2 + x_up_4  # tensor of shape (B, 4, 32, 32)

logits = F.interpolate(x_combined, scale_factor=2, mode='bilinear',
                       align_corners=True)  # tensor of shape (B, 2, 64, 64)

return logits
```

(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?

- The network is designed to predict grasp success using both local and global information. Local information is essential for determining the precise gripper position and orientation as well as the geometry, texture, and other aspects of the object that are important for a successful grasp. The network can evaluate the friction and stability of the grasp and understand the locations of contact between the gripper and the object using this information.

  Global information, on the other hand, is not strictly required for grasp prediction in this specific task, as the network focuses on predicting grasps based on top-down RGB images of the scene. However, it can still offer helpful context by showing things like how they are organized overall, how they relate to one another, and any potential occlusions or impediments that might get in the way of a grasp.

  Global information might be more important in other tasks. For example, in a robotic navigation task, global information would be necessary to understand the overall environment and plan an optimal path. This could include the locations of obstacles, the layout of the room, and the destination point. In such tasks, it is crucial to have a broader understanding of the scene to make effective decisions and plan safe, efficient paths.

Q3. 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.

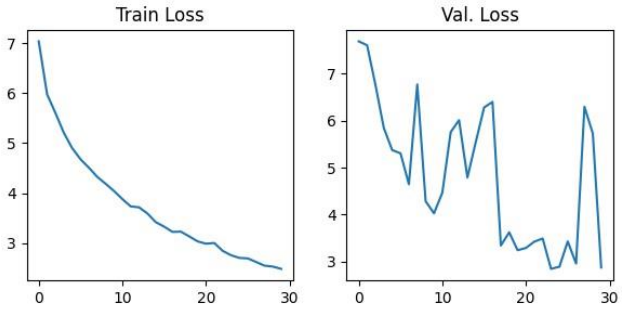


Figure 3: Example loss curves.

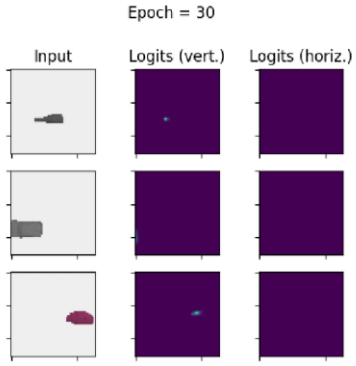
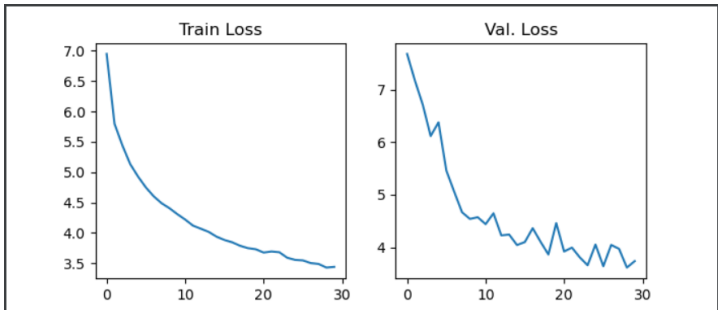


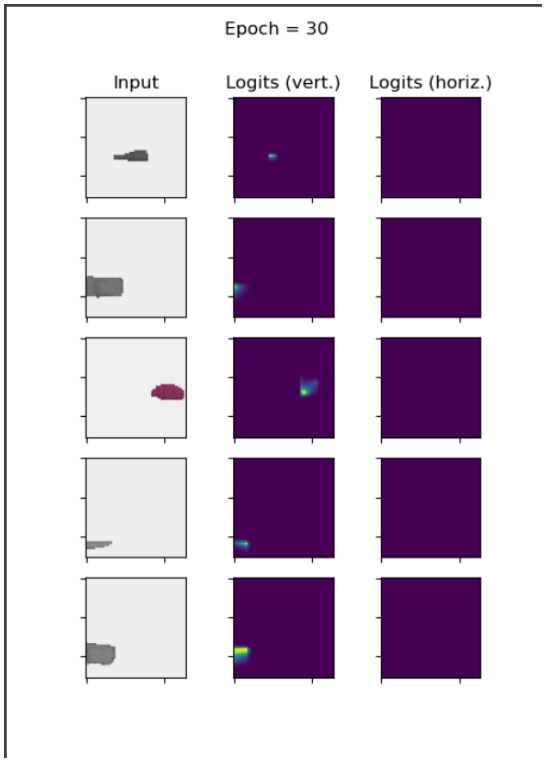
Figure 4: Example predictions.

```
PS C:\Users\Notsky\OneDrive\Surface Pro 6\Documents\Notsky\Education\Masters\Northeastern University\5th Semester - Spring 2023\5335 - Robotics\Week 11\hw6> python trainer.py
loss=3.74: 100%|| 30/30 [12:12<00:00, 24.43s/it]
PS C:\Users\Notsky\OneDrive\Surface Pro 6\Documents\Notsky\Education\Masters\Northeastern University\5th Semester - Spring 2023\5335 - Robotics\Week 11\hw6>
```

<u>After Training:</u>



<u>My Loss Curves</u>



<u>My Predictions</u>

- EXTRA CREDIT! - The validation loss curve may appear noisy, which could be due to several reasons:
    - Small dataset: If the dataset used for validation is small, the validation loss curve may exhibit significant noise. This is because the small dataset may not be representative enough to provide a stable estimate of the model's generalization performance.
        - Our training set is based on 800 successful grasps and validation set is 200
        - Given the values above in the context of deep learning the dataset is relatively small
    - High learning rate: If the learning rate is too high, the model may struggle to converge, causing the validation loss curve to oscillate. This is because the model overshoots the optimal parameter values and ends up oscillating around them.
        - Learning rate (LR) = 5e-4 or 0.0005
        - Base learning rate for Adam optimizer is 0.001
        - Our learning rate is lower than the base learning rate for Adam so it is considered a moderate learning rate
    - Dataset quality: The presence of outliers, noise, or inconsistencies in the dataset can cause the validation loss curve to be noisy. The model may struggle to generalize well on such noisy data.
    - Model complexity: If the model is too complex, it might overfit the training data and not generalize well to the validation data, resulting in a noisy validation loss curve.
        - The architecture we are using is MobileUNet which is a custom implementation of a U-Net style network with a pretrained MobileNetv3 backbone.
        - From our custom layers (I posted a pic above in Q2a based on the given Figure 2), it seems that it is not overly complex.
    - Stochastic gradient descent: When using stochastic gradient descent (SGD) or a variant (e.g., Adam, RMSprop), the optimization process is inherently noisy, as it relies on random mini-batches of data. This could lead to noisy validation loss curves.

(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)?

- The validation loss simply gauges the discrepancy between the network's predictions and the ground truth labels, so it is not a perfect indicator of how well the network can anticipate grasps. Yet, there may be more than one appropriate grasp site for a given object because the problem of grab success prediction is complex and spatially continuous.

  Even if a network predicts a valid grasp site that differs significantly from the ground truth label, the grasp will still be successful. Even though the anticipated grip is practically valid in these situations, the validation loss may still be quite significant because of the discrepancy between the forecast and the actual data. As a result, the network's forecast would be penalized even though it has discovered a workable grip.

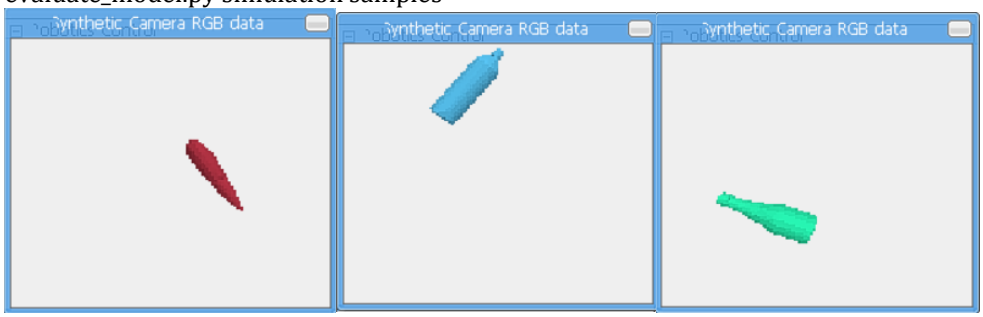
  Furthermore, the validation loss does not account for other elements that affect a grip's success, such as the quality of the grasp, its stability, or its resistance to perturbations. These elements might be essential in real-world situations, but the validation loss might not correctly reflect them.

  As a result, even if the validation loss can shed light on the network's effectiveness, it shouldn't be regarded as the only metric of grasp prediction performance. To gain a deeper knowledge of the network's grasp prediction skills, it is crucial to combine it with additional evaluation metrics or practical experiments.

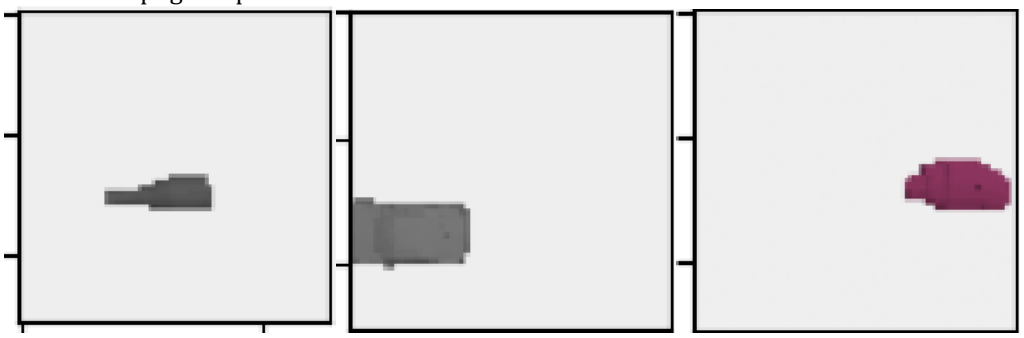
(c) To get a better sense of the model performance, we can evaluate it directly in the simulator where the data was generated. Run the command $python evaluate model.py to see the model perform 50 grasp attempts. What success rate is achieved (report the final estimate which is averaged over all 50 attempts)? Do you notice anything different about the object placements here (compared to what you see in 'predictions.png')?

```
pybullet build time: Apr  5 2023 15:57:07
starting thread 0
started testThreads thread 0 with threadHandle 000000000000031C
argc=2
argv[0] = --unused
argv[1] = --start_demo_name=Physics Server
ExampleBrowserThreadFunc started
Thread TERMINATED
finished
numActiveThreads = 0
btShutDownExampleBrowser stopping threads
b3Printf: Selected demo: Physics Server
starting thread 0
started MotionThreads thread 0 with threadHandle 00000000000008A8
MotionThreadFunc thread started
Success rate = 34.0%: 100%|████████████████████████████████████████████████████████████████████████████████| 50/50 [00:16<00:00,  2.94it/s]
Final success rate: 34.0%
numActiveThreads = 0
stopping threads
Thread with taskId 0 with handle 00000000000008A8 exiting
Thread TERMINATED
finished
numActiveThreads = 0
btShutDownExampleBrowser stopping threads
Thread with taskId 0 with handle 00000000000005B4 exiting
Thread TERMINATED
```

- Success Rate = 34% over 50 grasp attempts
- evaluate_model.py simulation samples



- Predictions.png samples



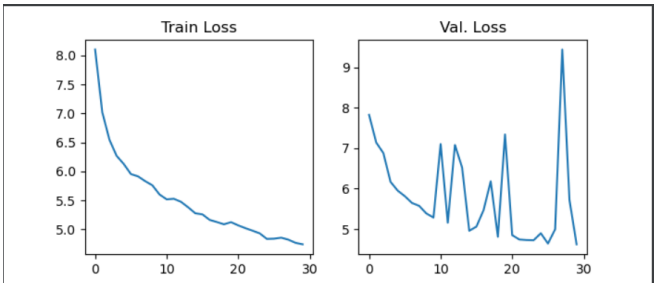
- Looking at the pic, we can see that in the simulations run by evaluations_model.py, the orientations have randomized angles while in predictions.png they are all oriented horizontally.

Q4. 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.



- Success Rate = 36%

(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.

- Image translations: We can randomly translate the image in both the horizontal and vertical directions. This can be achieved using the torchvision.transforms.RandomAffine transform in PyTorch, which allows us to perform random affine transformations on the image. We can set the translate parameter to a tuple of values specifying the maximum translations in the x and y directions.

- Pixel location: We can randomly shift the pixel location of the grasp in both the horizontal and vertical directions. This can be achieved by adding a random value to the current pixel location. We can set the range of the random value to be a small fraction of the image size, such as +/- 5% of the image width and height.

- Gripper rotation: In addition to the rotations by 0, 90, 180, or 270 degrees, we can also randomly rotate the gripper by a smaller angle. This can be achieved by generating a random angle between, say, -10 and 10 degrees, and adding it to the current angle. We can also randomly flip the gripper horizontally or vertically.