# Homework 4

## 2a

The problem with one-hot pixel as the target instead of using Gaussian Scoremap is that model predictions that are close to the actual label coordinates are not rewarded at all, i.e predictions that are off by one pixel and predictions that are way off incur the same loss. Therefore, it would be harder for the model to converge.

## 2b

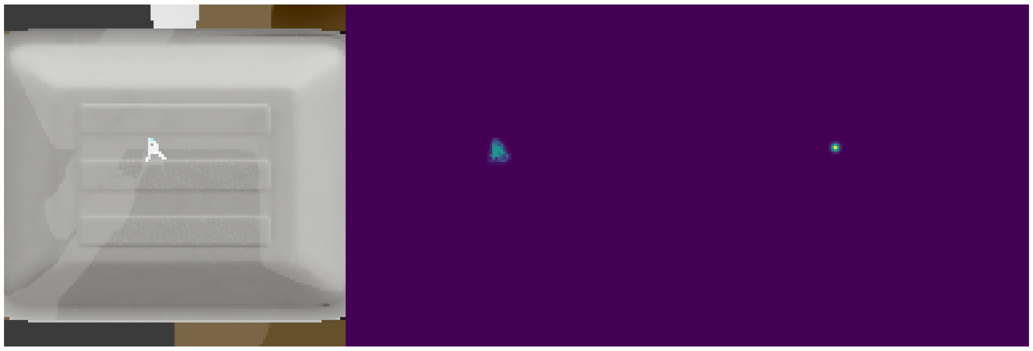
*self.aug_pipeline* aims to make the model more robust by creating some variances in the input dataset. Specifically, there is a 70 percent probability that the images inputted to the dataset would be translated vertically or horizontally by 0 to 20 percent of the total image size. Moreover, the images can also be rotated from -22.5 degrees up to 22.5 degrees

## 2c

If we are using *nn.BCELoss*, The *nn.Module* expects the functions input to be a value between 0. and 1., i.e activation layer of the last layer of the Neural Network should be a sigmoid function. Instead of using this function, Pytorch created *nn.BCEWithLogitsLoss* that combines these two steps together, creating a numerical stability as we now can use the *log-exp-sum* trick. Thus, we no longer need sigmoid activation function in our output layer if we are using this loss function.

## 2d

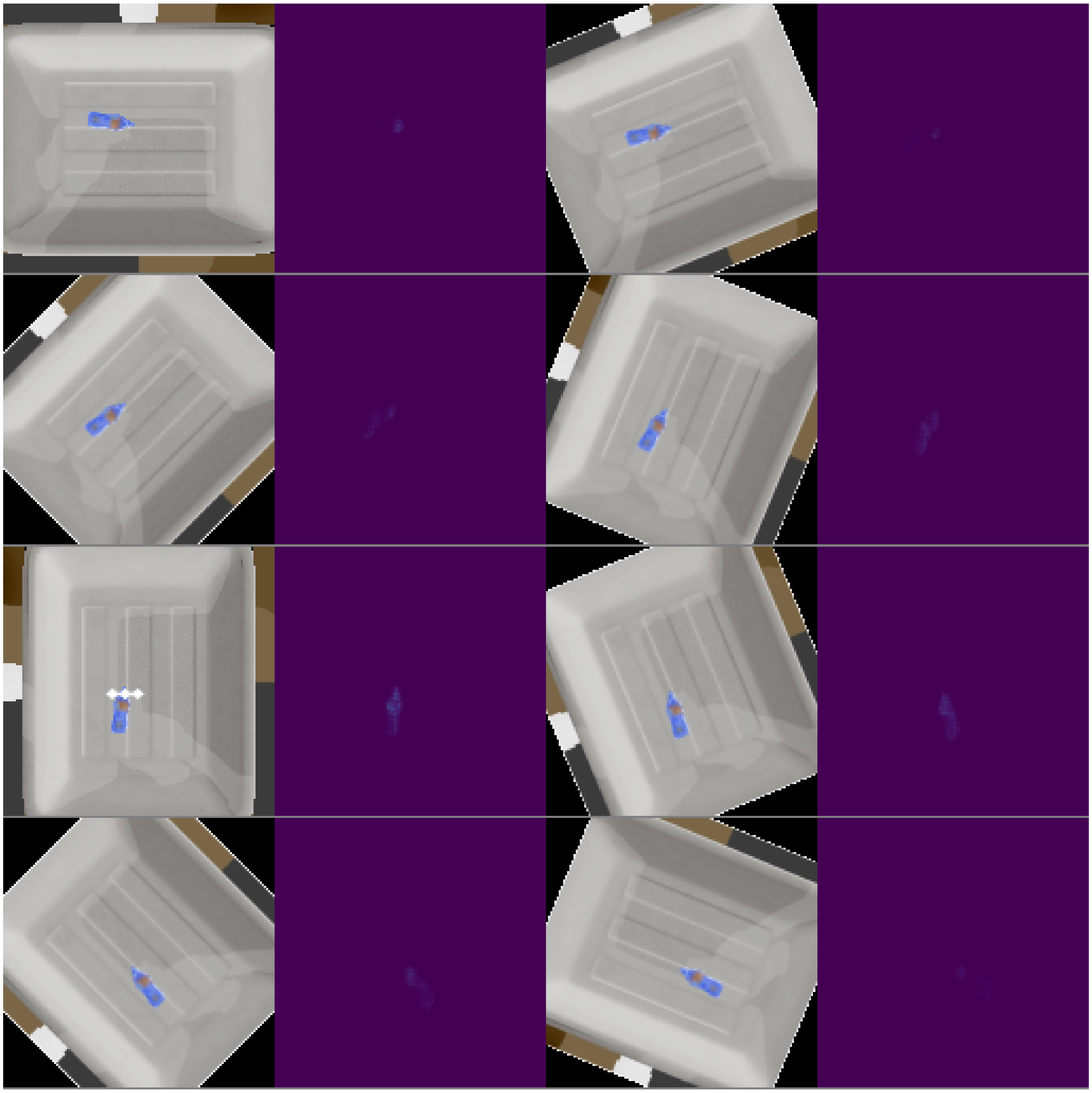
Train Loss : 0.0091 Test Loss : 0.0096



## 2f

Link: https://drive.google.com/file/d/1qjGEVEVsgpNRryCHVLNfEbrbNHpNppwq/view?usp=sharin

Success Rate : 80 percent



## 2g

The method is very sample efficient as the Data Augmentation helps the model to understand the different affordances for all of the possible angles. The Gaussian score-map target also penalizes less if the model is close to the actual target, helping it to converge faster. Lastly, the model receive 8 inputs, for all of the possible grasp angle, allowing it to consider the best affordances out of all angle.

Item Left : 2

Link : https://drive.google.com/file/d/1j36O9lTmoKy1aa-9XoqrzkKpViUWRzez/view?usp=sharing

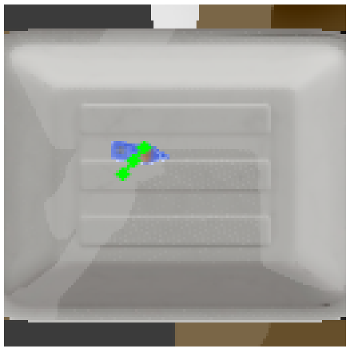
## 3a

Train Loss : 0.0403 Test Loss : 0.0096

## 3b

Success Rate : 13.3 percent

Link : https://drive.google.com/file/d/1T-J4nZrc7Z6KxunjvKbvKjzCxpKDrDpw/view?usp=sharing



## 3c

The model performs worse because it does not encode the relationship between the coordinates (x and y), as well as the coordinate and the angle with regards to its affordance. The affordances of prediction with accurate coordinate but slightly inaccurate angle might be very low, even though the loss function under action regression might be small. Similarly, a prediction with slight inaccuracy of x and y prediction might yield high affordance, and inaccuracy of x, and accurate prediction of y might yield low affordance (imagine a long, horizontal item). However, in these cases, action regression will give the same loss for both scenario, rendering the output prediction to be inaccurate.

Items Left : 13

Link : https://drive.google.com/file/d/1OvuOgO7xA38Trr4_czRIkRRCviiSbCzi/view?usp=sharing

## 4

In performing this project, I have noticed various problem that motivates me to make an improvement to the algorithm:

- It is often that the robot is stuck. If there are no changes in the environment, the image input to the model will stay the same, and thus the maximum affordance coordinate and angle selected will remain the same. Therefore, if the robot failed to pick up an item, then this will result in indefinite loop
- The model does not perform well in picking up new items. For example, Compared to the items in the training set, scissors are much flatter, and thus the angle in which it picks up the item is far more important compared to mustard bottle, for example
- The model is not regularized

Therefore, there are three improvement that I've made in the model in which explained below

### Self-Supervised Learning

The goal of self-supervised learning is to let the model collect the training data on its own. In other words, we require an image of the environment, as well as a location (coord, angle), in which a grasping can be completed. However, letting the model choose a completely random coordinate and angle will result in a very high probability of failure.

Therefore, I've decided to use the affordance model trained earlier as the base model to perform the self-supervised learning. Nevertheless, I've purposefully chosen not the best affordance, but randomize from N-th best affordance location, in order to perform target augmentation, to allow the model to learn different possible grasping of an object

This is trained on both training and test dataset

### Change in Network Architecture

Similar to the original U-Net architecture, I've decided to introduce Batch Normalization and Dropout to the Neural Network. Batch Normalization aims to normalize each of the layer inputs, such that internal covariate shift does not occur. This helps the model to converge faster. Furthermore, Dropout is also introduced such that the model does not overfit.

### Randomize Action

In order for the network not to be stuck in suboptimal action, The robot choose randomly one action out of N-th best affordances. This ensure the model to choose other actions if it fails to grasp an object

However, upon experiment, it seems that this model have not been able to improve the performance of the original model. Various hypothesis on why this model underperform are:

- Unlike the manual label, where the grasping location is meticulously chosen to be optimal (centered, away from wall, parallel to the grasper), the self-supervised learning might choose suboptimal action that still might result to a successful grasp, and thus resulting in suboptimal ground truth. In order to solve this problem, a higher number of training data is needed, similar to the technique discussed in this paper : (https://arxiv.org/abs/1509.06825)
- The self-supervised learning training data is trained using the headless mode. Upon checking the Ed discussion, it seems that using the headless mode is discouraged. I am unable to find enough time to retrain the self-supervised learning algorithm.
- The Dropout parameter have not been tuned. Due to the small size of the Neural Network, it is possible that the parameter is too huge

- The Random action parameter have not been tuned. Choosing random action parameter $N$ that is too small might result in robot "stuck" choosing the same action, while $N$ that is too large might result in suboptimal action.

Link : https://drive.google.com/file/d/1XrBkz5xjDq3hTemJEJfnc4XjJ5SmWDk7/view?usp=sharing

Items Left : 5