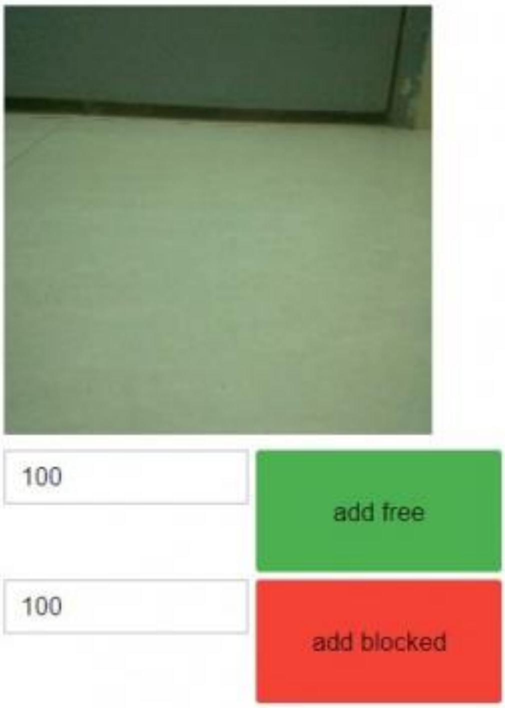
Collision Avoidance

In this example, we'll collect an image classification dataset that will be used to help keep JetBot safe! We'll teach JetBot to detect two scenarios free and blocked. We'll use this AI classifier to prevent JetBot from entering dangerous territory.

Step 1: Collect data on JetBot

- Access JetBot by going to https://<jetbot_ip_address>:8888, navigate to ~/Notebooks/collision_avoidance/.
- Open data_collection.ipynb file and following notebook.
- After running the program, the interface as shown in the figure appears, put the car in a different position, and click "add free" if there is no obstacle in front of the car. If there is an obstacle in front of the car, please click "add blocked".
- The captured pictures will be saved in the dataset folder, and as many pictures of various situations as possible will be taken. You can try different orientations, brightness, object or collision types (walls, ledges, etc.), and can try untextured floors/objects (patterned, smooth, glass, etc.).

```
[6]: display(image)
     display(widgets.HBox([free_count, free_button]))
     display(widgets.HBox([blocked_count, blocked_button]))
```



- The more scene data the car collects, the better the obstacle avoidance effect will be. Therefore, it is very important to obtain as much different data as possible for the obstacle avoidance effect. Generally, at least 100 pictures are required for each situation.

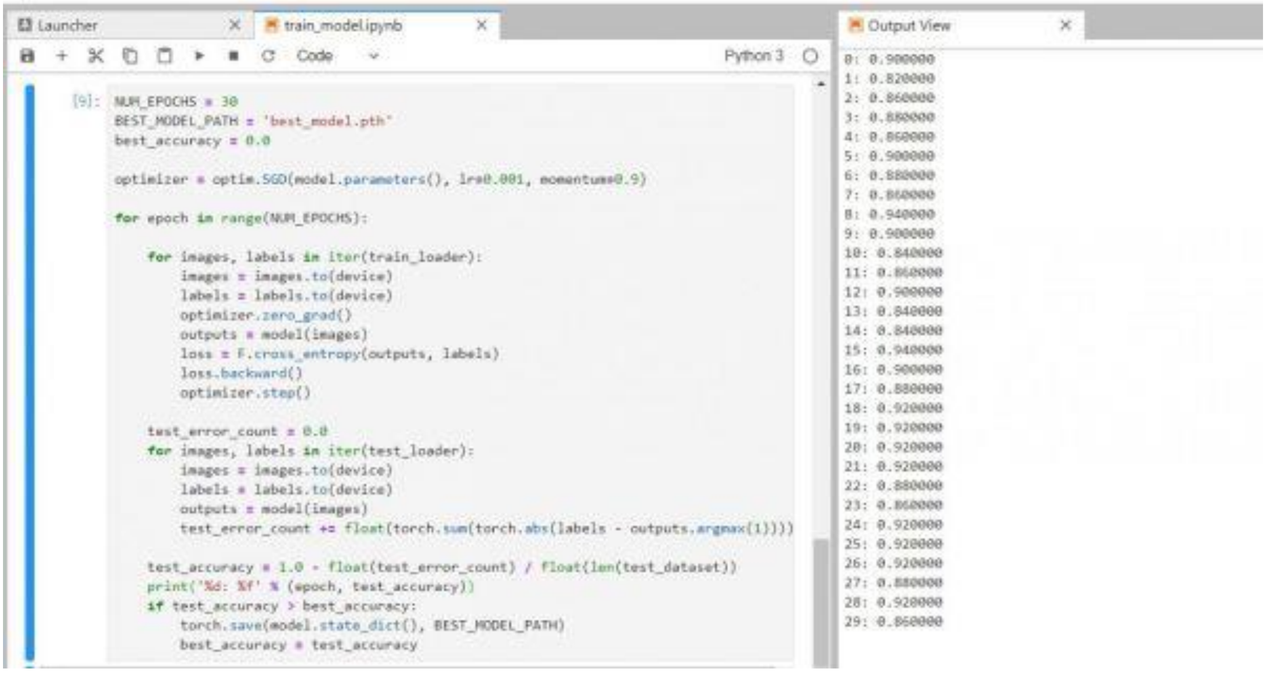
- Finally, run the program to package the pictures. After packaging, a dataset.zip compressed file will appear in the current directory.

```
[7]: !zip -r -q dataset.zip dataset
```

Step 2. Train neural network

- Access JetBot by going to https://<jetbot_ip_address>:8888, navigate to ~/Notebooks/collision_avoidance/
- Open and follow the tain_model.ipynb notebook.
- If you already have the dataset.zip file you just compressed, you do not need to run this statement to decompress it, otherwise you will be prompted to overwrite the existing file.

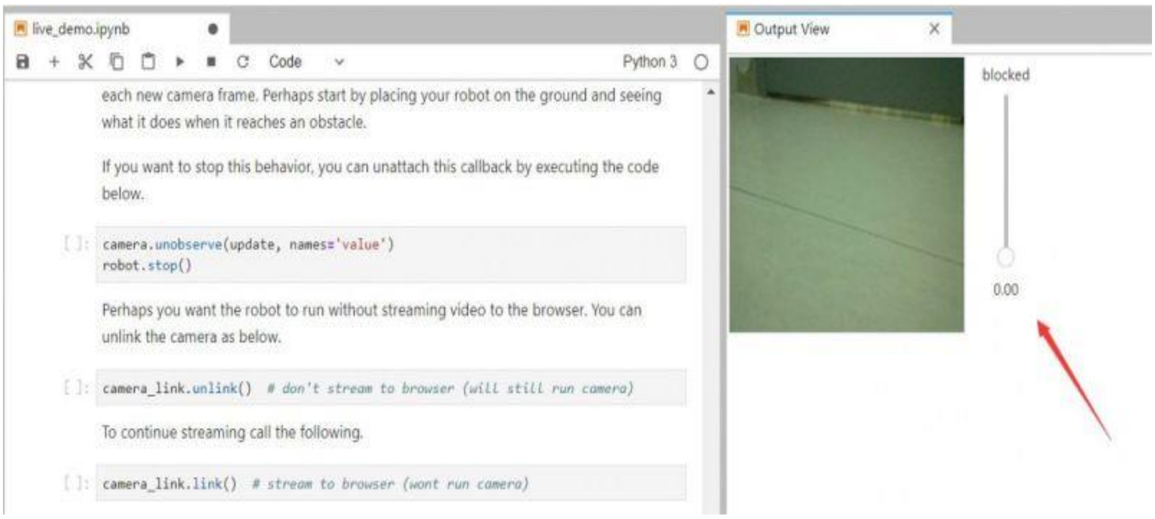
- Finally, run the program to train the neural network, and the running time is relatively long. After the training is completed, a best_mode.pth file will appear in the current directory.



- When the training of the model is going on then the value of the accuracy menas the value 1.00000 menas it is the best accuracy also 0.9 is also considerd as the best accuracy. From 0.5 to 0.0 considerd as worst accuracy

Step 3. Automatic Obstacle Avoiding

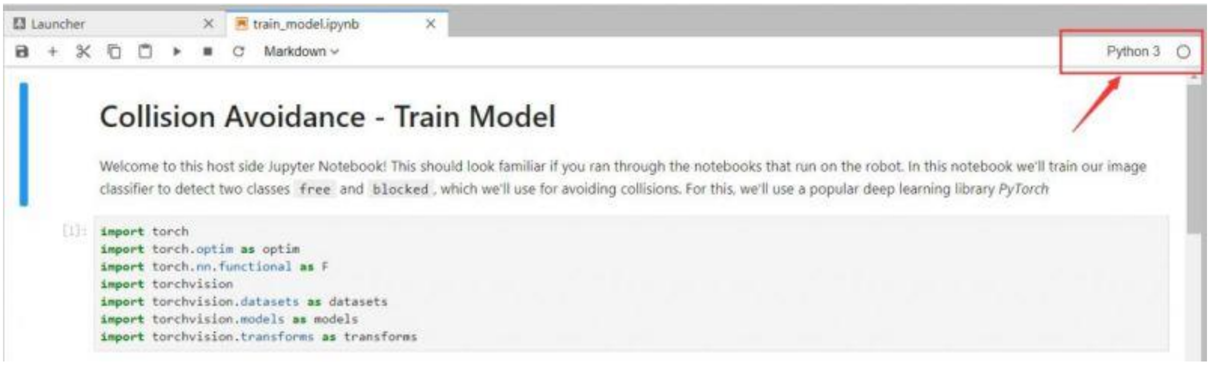
- Access JetBot by going to https://<jetbot_ip_address>:8888, navigate to ~/Notebooks/collision_avoidance/
- Open and follow the live_demo.ipynb notebook.
- After running the program, the camera live image and a slider are displayed. Intermodulation represents the probability of encountering an obstacle, 0.00 means that there is no obstacle ahead, and 1.00 means that the obstacle ahead needs to be turned to avoid.



- Here, adjust the speed a little to avoid hitting the obstacles too fast. If obstacle avoidance cannot be achieved in some places, it is recommended to collect more data.

```
[ ]: import torch.nn.functional as F
     import time

     def update(change):
         global blocked_slider, robot
         x = change['new']
         x = preprocess(x)
         y = model(x)

         # we apply the `softmax` function to normalize the output vector so it sums to 1 (which makes it a probabilit
         y = F.softmax(y, dim=1)

         prob_blocked = float(y.flatten()[0])

         blocked_slider.value = prob_blocked

         if prob_blocked < 0.5:
             robot.forward(0.2)
         else:
             robot.left(0.1)

         time.sleep(0.001)

     update({'new': camera.value})  # we call the function once to intialize
```

- [Important Note] When the Jetbot gets start then execute the next cell as it connects the camera so it will avoid the obstacles otherwise it will not stop as the camera instance needs to connect then you can see the 1.00 to the blocked image and 0.00 to free image
- [Note] Some statements may take a long time to run. There is a program running promptly in the upper right corner of JupyterLab. When the small dot is black, it means the program is running, and white means it is idle.



The model is trained by using red tape so that it will recognize the obstacle as a red tape

If the model training is more then you will get more accuracy.