CS8803 - Project 2

Convolutional Neural Networks

Jonas Loy, Garrett Mallory

November 2016

1 Terrain Traversability

1.1 Data Collection

The first model is based on images gathered around the center island of the lakeExample.ttt scene. Special emphasis was made on viewing the sky box. The other non-shore obstruction in the map, the flag pole, was not viewable by our agent due to the shore. The traversable images, defined as having a z values of ≤ 0.5, were a mixture of open landscape and shore line. For various experiments, either 2000 or 6000 images were collected, split evenly between traversable and intraversable images.

1.2 Model Training and Performance

In the images below, red indicates traversability and green intraversability. The first set of results examines model performance on the 2k images dataset for various iteration counts. In particular, the open horizon above the lake is well separated from the sky as can be seen in Figure 1.

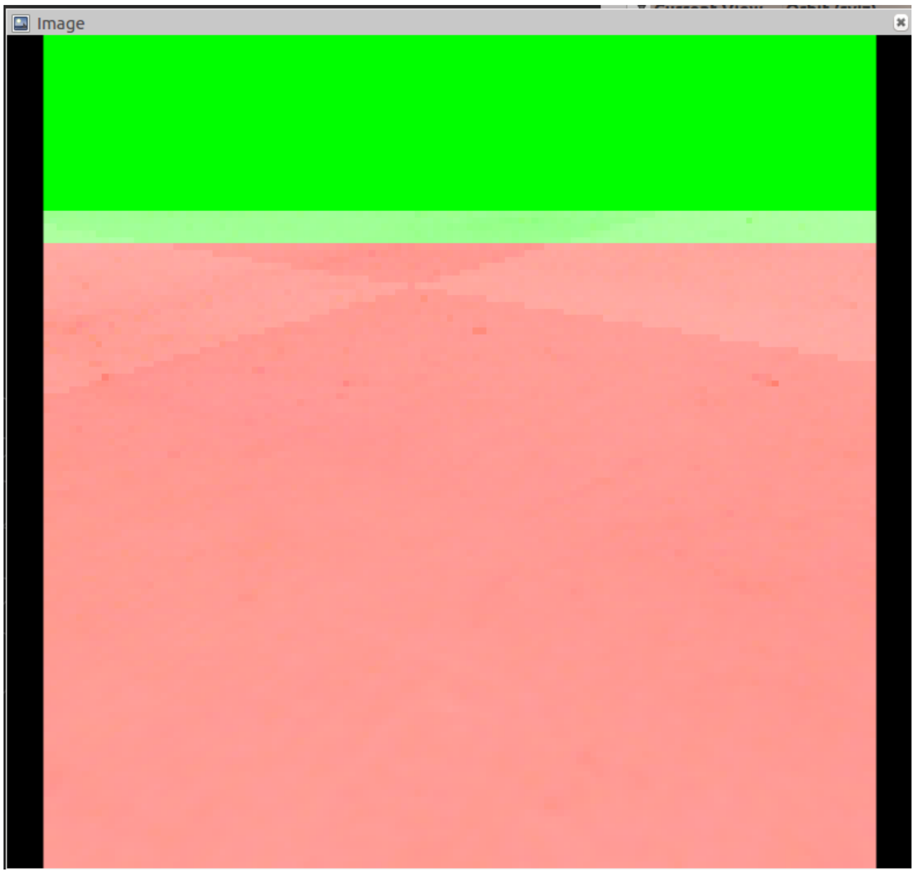


Figure 1: Horizon with training set size = 2k, iterations = 1k

More interesting, is how well the classifier handles the box and the shoreline.

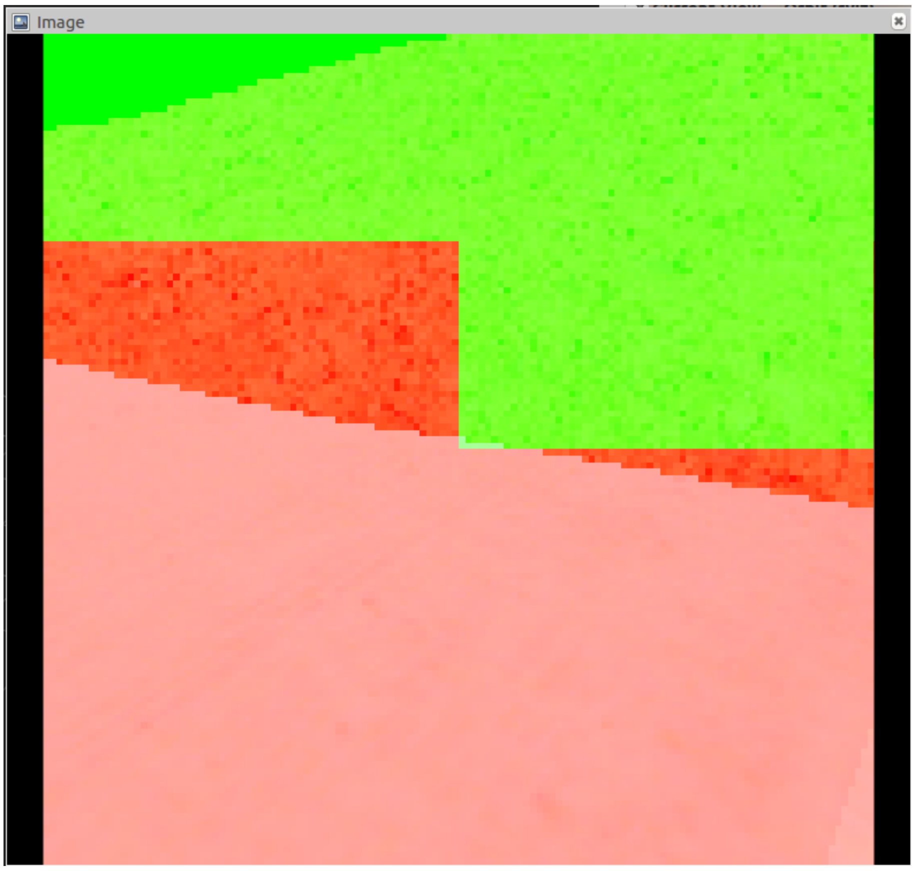


Figure 2: Shoreline with training set size = 2k, iterations = 1k

The shore is well modeled, but the classifier struggles some with the box. This improves when a 10k iteration model is used. A comparison of the two performances can be seen below.

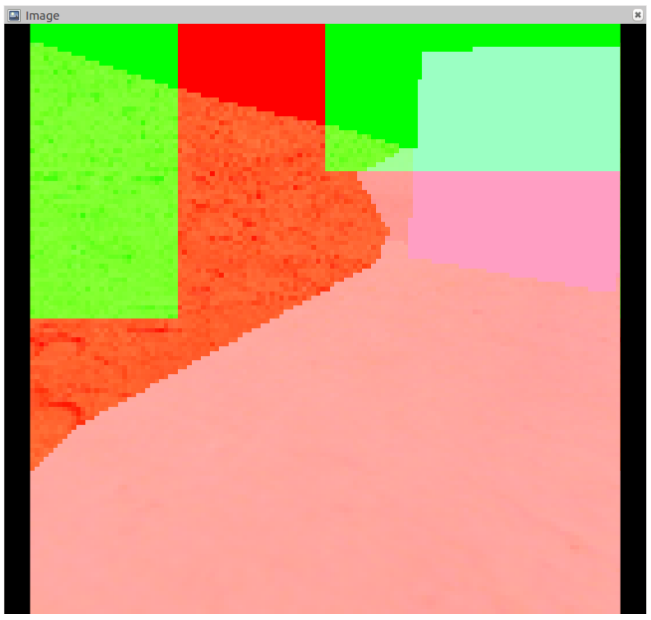


Figure 3: Box view with iterations = 1k.

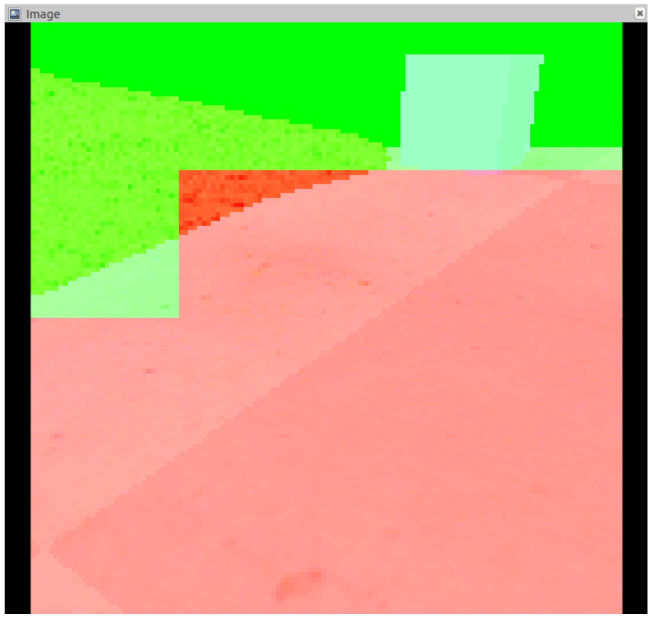


Figure 4: Box View with iterations = 10k.

1.3 Comparison to Higher Iteration Count

At first, comparisons of different dataset's performance with 1k, 10k and 15k iterations indicated overfitting of the data at higher iteration counts resulting in a classification of everything as traversable. In the 15k iteration sample on one dataset, there were only 2 cases where sections would be labeled intraversable: images looking squarely at the sky box with the box filling it's view and when the robot pushes up on shore and subsequently tilts back and gets an image of the sky. Figure 5 shows what the box looks like when trained over 15k iterations of a dataset of size 2k.

However, in closer examination of the learning process, it became clear that higher iterations will not overfit but instead more and more closely approximate a solution. This is due to CaffeeNet's built in processes to prevent overfitting. As more data was gathered, it became evident that the data collection process was the more impactful aspect of the final model. Some models perform better than others at all iteration counts due to having a more balanced, more complete initial input.

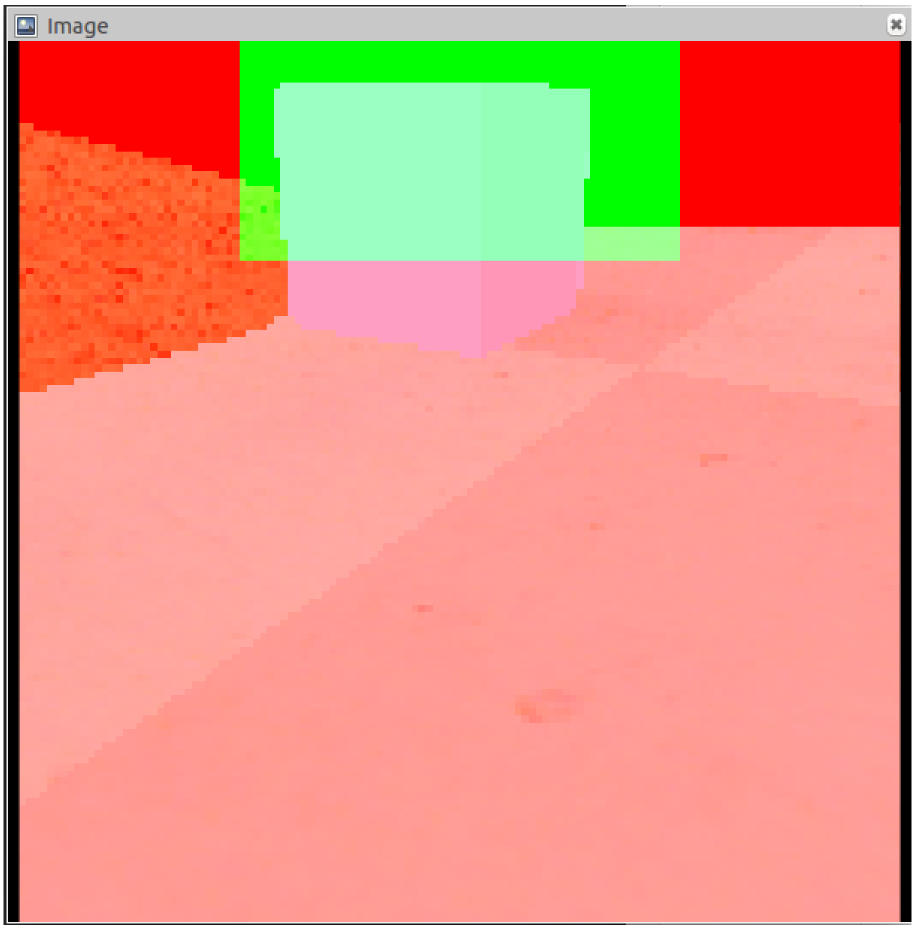
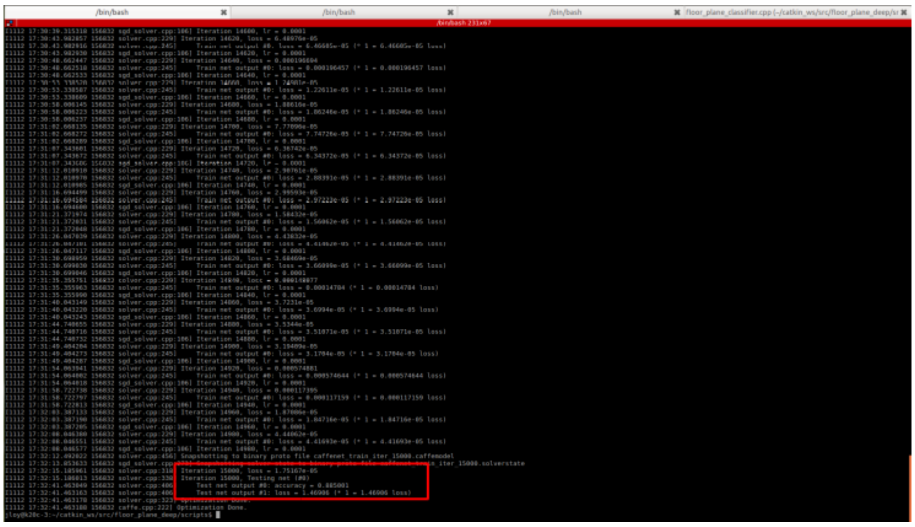


Figure 5: Decreased box classification performance with 15k iterations

1.4 Comparison to Higher Training Set Size

It was thought that having 3000 images of each type of terrain (for a total of 6000 images) would lead to better performance but this has not been the case. In fact, the label is always the same: traversable. Looking at the box and the sky still produces the same label. This goes against intuition that larger dataset sizes generate better results, but it is possible that while trying to gather novel images from all around the box, many of the images were repeated with only slight variations. If this occurred (a near certainty given the difficulty of driving the robot and speed at which images are gathered), the model could have been biased to particular types of images, leading to weaker generalization capability and weaker performance. Again, the collection process makes or breaks the final model.

This becomes especially evident when the performance of the model on the testing set is considered. Below is an image showning 88% accuracy on the test set which was gather in the same philospohy as the training set. However, this 88% does not reflect real world accuracy, which as we can tell from the declining images above, can be much lower. This accuracy metric was fairly consistent across all of the datasets we examined, yet real world performance varied greatly.


/bin/bash
/bin/bash
/bin/bash
floor_plane_classifier.cpp (~/catkin_ws/src/floor_plane_deep/sr
/bin/bash 231x67

```
I1112 17:30:39.315318 156832 sgd_solver.cpp:106] Iteration 14600, lr = 0.0001
I1112 17:30:43.982857 156832 solver.cpp:229] Iteration 14620, loss = 6.48976e-05
I1112 17:30:43.982916 156832 solver.cpp:245]     Train net output #0: loss = 6.46605e-05 (* 1 = 6.46605e-05 loss)
I1112 17:30:43.982930 156832 sgd_solver.cpp:106] Iteration 14620, lr = 0.0001
I1112 17:30:48.662447 156832 solver.cpp:229] Iteration 14640, loss = 0.000196694
I1112 17:30:48.662510 156832 solver.cpp:245]     Train net output #0: loss = 0.000196457 (* 1 = 0.000196457 loss)
I1112 17:30:48.662533 156832 sgd_solver.cpp:106] Iteration 14640, lr = 0.0001
I1112 17:30:53.338520 156832 solver.cpp:229] Iteration 14660, loss = 1.24981e-05
I1112 17:30:53.338587 156832 solver.cpp:245]     Train net output #0: loss = 1.22611e-05 (* 1 = 1.22611e-05 loss)
I1112 17:30:53.338609 156832 sgd_solver.cpp:106] Iteration 14660, lr = 0.0001
I1112 17:30:58.008145 156832 solver.cpp:229] Iteration 14680, loss = 1.88616e-05
I1112 17:30:58.008223 156832 solver.cpp:245]     Train net output #0: loss = 1.86246e-05 (* 1 = 1.86246e-05 loss)
I1112 17:30:58.008237 156832 sgd_solver.cpp:106] Iteration 14680, lr = 0.0001
I1112 17:31:02.668135 156832 solver.cpp:229] Iteration 14700, loss = 7.77096e-05
I1112 17:31:02.668272 156832 solver.cpp:245]     Train net output #0: loss = 7.74726e-05 (* 1 = 7.74726e-05 loss)
I1112 17:31:02.668289 156832 sgd_solver.cpp:106] Iteration 14700, lr = 0.0001
I1112 17:31:07.343801 156832 solver.cpp:229] Iteration 14720, loss = 6.36742e-05
I1112 17:31:07.343872 156832 solver.cpp:245]     Train net output #0: loss = 6.34372e-05 (* 1 = 6.34372e-05 loss)
I1112 17:31:07.343606 156832 sgd_solver.cpp:106] Iteration 14720, lr = 0.0001
I1112 17:31:12.010910 156832 solver.cpp:229] Iteration 14740, loss = 2.90761e-05
I1112 17:31:12.010970 156832 solver.cpp:245]     Train net output #0: loss = 2.88391e-05 (* 1 = 2.88391e-05 loss)
I1112 17:31:12.010985 156832 sgd_solver.cpp:106] Iteration 14740, lr = 0.0001
I1112 17:31:16.694499 156832 solver.cpp:229] Iteration 14760, loss = 2.99593e-05
I1112 17:31:16.694584 156832 solver.cpp:245]     Train net output #0: loss = 2.97223e-05 (* 1 = 2.97223e-05 loss)
I1112 17:31:16.694600 156832 sgd_solver.cpp:106] Iteration 14760, lr = 0.0001
I1112 17:31:21.371974 156832 solver.cpp:229] Iteration 14780, loss = 1.58432e-05
I1112 17:31:21.372031 156832 solver.cpp:245]     Train net output #0: loss = 1.56062e-05 (* 1 = 1.56062e-05 loss)
I1112 17:31:21.372048 156832 sgd_solver.cpp:106] Iteration 14780, lr = 0.0001
I1112 17:31:26.047039 156832 solver.cpp:229] Iteration 14800, loss = 4.43832e-05
I1112 17:31:26.047101 156832 solver.cpp:245]     Train net output #0: loss = 4.41462e-05 (* 1 = 4.41462e-05 loss)
I1112 17:31:26.047117 156832 sgd_solver.cpp:106] Iteration 14800, lr = 0.0001
I1112 17:31:30.698959 156832 solver.cpp:229] Iteration 14820, loss = 3.68469e-05
I1112 17:31:30.699030 156832 solver.cpp:245]     Train net output #0: loss = 3.66099e-05 (* 1 = 3.66099e-05 loss)
I1112 17:31:30.699046 156832 sgd_solver.cpp:106] Iteration 14820, lr = 0.0001
I1112 17:31:35.355751 156832 solver.cpp:229] Iteration 14840, loss = 0.000148077
I1112 17:31:35.355963 156832 solver.cpp:245]     Train net output #0: loss = 0.00014784 (* 1 = 0.00014784 loss)
I1112 17:31:35.355990 156832 sgd_solver.cpp:106] Iteration 14840, lr = 0.0001
I1112 17:31:40.043149 156832 solver.cpp:229] Iteration 14860, loss = 3.7231e-05
I1112 17:31:40.043220 156832 solver.cpp:245]     Train net output #0: loss = 3.6994e-05 (* 1 = 3.6994e-05 loss)
I1112 17:31:40.043243 156832 sgd_solver.cpp:106] Iteration 14860, lr = 0.0001
I1112 17:31:44.740655 156832 solver.cpp:229] Iteration 14880, loss = 3.5344e-05
I1112 17:31:44.740716 156832 solver.cpp:245]     Train net output #0: loss = 3.51071e-05 (* 1 = 3.51071e-05 loss)
I1112 17:31:44.740732 156832 sgd_solver.cpp:106] Iteration 14880, lr = 0.0001
I1112 17:31:49.404204 156832 solver.cpp:229] Iteration 14900, loss = 3.19409e-05
I1112 17:31:49.404273 156832 solver.cpp:245]     Train net output #0: loss = 3.1704e-05 (* 1 = 3.1704e-05 loss)
I1112 17:31:49.404287 156832 sgd_solver.cpp:106] Iteration 14900, lr = 0.0001
I1112 17:31:54.063941 156832 solver.cpp:229] Iteration 14920, loss = 0.000574881
I1112 17:31:54.064002 156832 solver.cpp:245]     Train net output #0: loss = 0.000574644 (* 1 = 0.000574644 loss)
I1112 17:31:54.064018 156832 sgd_solver.cpp:106] Iteration 14920, lr = 0.0001
I1112 17:31:58.722738 156832 solver.cpp:229] Iteration 14940, loss = 0.000117395
I1112 17:31:58.722797 156832 solver.cpp:245]     Train net output #0: loss = 0.000117159 (* 1 = 0.000117159 loss)
I1112 17:31:58.722813 156832 sgd_solver.cpp:106] Iteration 14940, lr = 0.0001
I1112 17:32:03.387133 156832 solver.cpp:229] Iteration 14960, loss = 1.87086e-05
I1112 17:32:03.387190 156832 solver.cpp:245]     Train net output #0: loss = 1.84716e-05 (* 1 = 1.84716e-05 loss)
I1112 17:32:03.387205 156832 sgd_solver.cpp:106] Iteration 14960, lr = 0.0001
I1112 17:32:08.046380 156832 solver.cpp:229] Iteration 14980, loss = 4.44062e-05
I1112 17:32:08.046551 156832 solver.cpp:245]     Train net output #0: loss = 4.41693e-05 (* 1 = 4.41693e-05 loss)
I1112 17:32:08.046577 156832 sgd_solver.cpp:106] Iteration 14980, lr = 0.0001
I1112 17:32:12.492022 156832 solver.cpp:456] Snapshotting to binary proto file caffenet_train_iter_15000.caffemodel
I1112 17:32:13.853633 156832 sgd_solver.cpp:273] Snapshotting solver state to binary proto file caffenet_train_iter_15000.solverstate
I1112 17:32:15.185961 156832 solver.cpp:310] Iteration 15000, loss = 1.75167e-05
I1112 17:32:15.186013 156832 solver.cpp:330] Iteration 15000, Testing net (#0)
I1112 17:32:41.463049 156832 solver.cpp:406]     Test net output #0: accuracy = 0.885001
I1112 17:32:41.463163 156832 solver.cpp:406]     Test net output #1: loss = 1.46906 (* 1 = 1.46906 loss)
I1112 17:32:41.463170 156832 solver.cpp:323] Optimization Done.
I1112 17:32:41.463188 156832 caffe.cpp:222] Optimization Done.
jloy@k20c-3:~/catkin_ws/src/floor_plane_deep/scripts$ 
```

Figure 6: Model training performance & loss on test dataset

2 Shore Follower

2.1 Data Collection

Many, many iterations of data collection and training were attempted for the shore follower dataset. Some of the evolutionary stages were:

- The first dataset type has took disrupted observations while driving due to erratic controls from running two conflicting joystick nodes at once. It took a bidirectional approach to traveling around the lake (i.e. traveled clockwise and counterclockwise).
- A second dataset has the erratic controls fixed by only running one joystick. It also goes in two directions around the lake.
- A third dataset type has images taken from the outer rim of the lake and around the inner island (thereby enabling counterclockwise and clockwise motion respectively to help generate data more rapidly). It chooses just one direction to travel in, namely the direction that keeps the shore on the right hand side. Images of open water & approaching shore were included.
- Last, data was collected simply by going one direction around the lake, ignoring the inner island and open lake until all images types (left, straight, right) had full collections. This took substantially longer as moving one direction tends to be dominated by one turn type. This dataset type includes variations in how closely the robot keeps to the shore.

There are overall 4 types of data: 1) Looking at the horizon 2) Looking squarely at the shore 3) Traveling parallel with the lake on your left 4) Traveling parallel with the lake on your right.

Type 1 exists so that if the robot is ever lost away from the lake shore, it will continue straight until it hits the shore again. This helps increase robot robustness and prevent idling in the open water. Type 2 begins by looking squarely at the shore and then turning from being perpendicular to parallel to the shore so the robot can follow the shoreline. When both types 3 & 4 exist, the robot alternates between looking left and right. If only of these types exist, it will always turn to one side in agreement with the side the shore needs to be on to collect data.

As we amended our data collection scheme, we vary which types of data are present and in what proportions. Initially, all 4 data types are used but by the final dataset, poor performance of early datasets causes us to use only type 3 or 4 (but not both).

When following the shore, we strive to maintain a small but reasonable distance and to make images consistent across the dataset. Below is an image of our collection strategy. We balance keeping the distant shore horizon in the middle 1/3 of the top of the image (between the green bars) and the nearer horizon on one side of the lower part of the screen (between one of the sets of blue bars). Either will do so long as the data is consistent. In our case, we keep the near shore between the blue bars on the lower right as we keep the shore on the right as much as possible. This strategy allows stable collection and avoidance of collision with the shore. Different datasets take different approaches to the shore distance - some will follow every crevice, generating highly nuanced turning data, while others will continue straight ahead unless finding it necessary to turn to avoid collision with the shore, generating steady data collection & more generic images of the shore on the side. The Choppy dataset uses only 1 control at a time to collect data. It alternates between going straight and turning to avoid giving more than 1 joystick input at once. The datasets labeled "Final" attempts to add more noise to the process by maintaining a wider range around the ideal distance and gives more examples of the robot approaching the idea line.

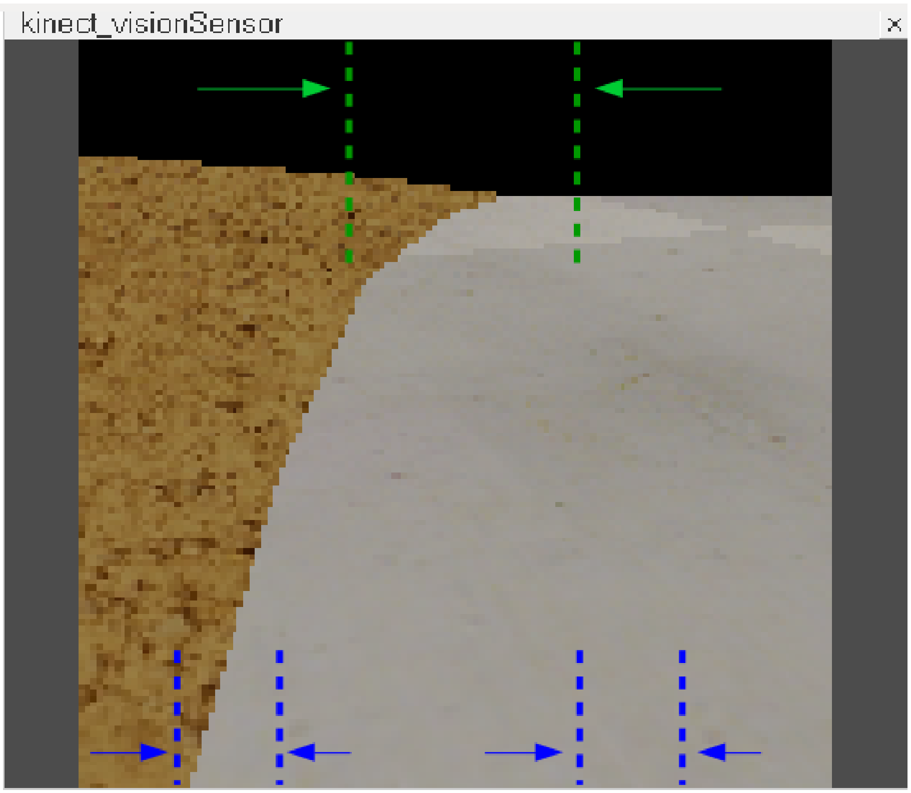


Figure 7: Image showing collection strategy

2.2 Model Training and Performance

Models were initially trained with 10k iterations and snapshots were taken at every 2k iterations. However, it quickly became clear that this may not be sufficient for convergence and so later models were run with 100-150k iterations with snapshots taken at every 10k iterations.

There were a variety of results loosely correlated with the collection strategy. At first, we found that the neural net's decision probabilities were frozen on particular values representing the proportions of data labels present (left, straight, and right all equal 33%). In one case, the medium sized one directional dataset, the dataset possessed 50% Straight labels and 25% Left and Right labels. The model learned that for all inputs, it should go straight with probability 48% and left and right with probabilities 28% and 28% each.

Other models would learn to have boosted confidence in just one or two directions while virtually ignoring the remainder, either by assigning a constant or zero value to the other directions or by making the input images needed to trigger them very precise so as to be next to impossible to generate. This typically manifested itself in agents who would turn left when the shore was on the right and turn right when data showed the shore on the left. This makes some intuitive sense as the agent is dominated by turns to one side, but it is unclear why capping the input images for each type did not resolve this issue. Unless specifically intended, all datasets include 3000 images, 1000 of each direction. Also, considering the data is shuffled when creating the lmdb files, the model should not be biased to whichever collection type finishes first.

Overall, the best performance was seen in the first dataset taken in one direction around the lake (Dataset = Data_oneDirection_trial01, iterations = 100k). While it classifies straight and left images fairly well, it's drawback is that it fails to identify situations where it should go to the right to stay close to the shore on it's right hand side. It's not immediately obvious why the CNN would learn to discount one of the label outputs in favor of the first two. The end result is that the robot tends to wonder off into the lake after some time of following the shore. This is accelerated when the robot needs to cut back to the right to round a bend in the shoreline. Below is an example of what the trajectory might look like with this model. We can see the robot adheres to the shore before getting ”spooked” and turning toward the open water.

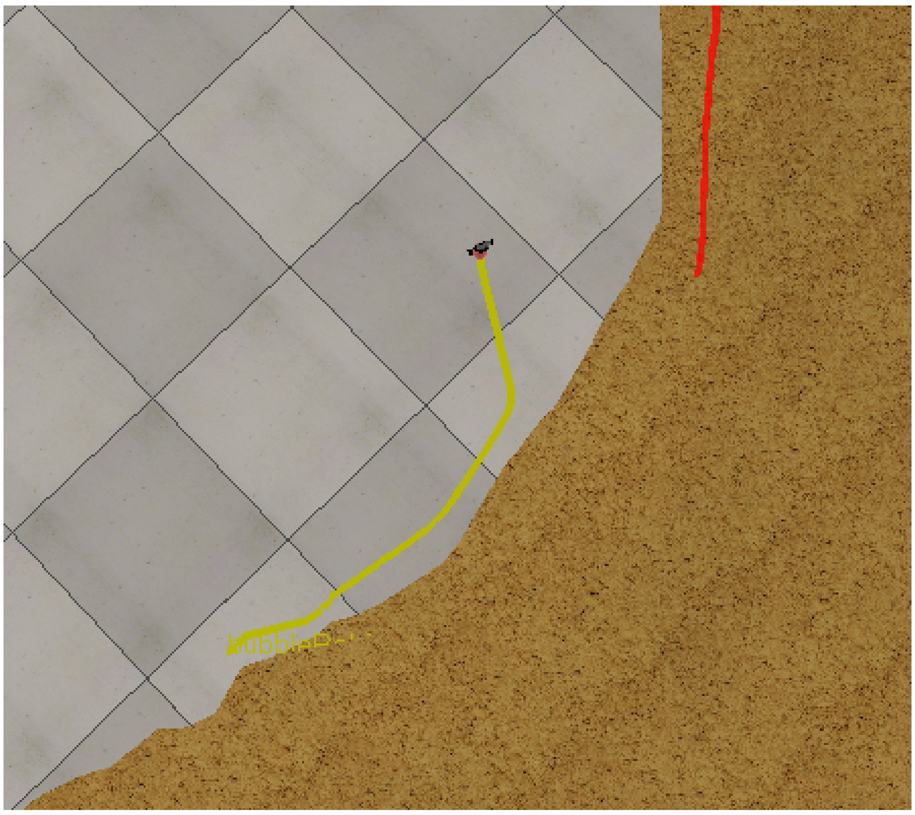


Figure 8: One Direction model performance at 100k training iterations.

3 Conclusion

Many machine learning algorithms are notorious for having many parameters to tune or for being difficult to monitor. This is especially true for convolutional neural nets. The hidden layers add mystery to the inner workings of the net and make it next to impossible to debug outputs given particular inputs. This, coupled with the sheer variety of inputs possible mean that the results of the net fluctuate greatly between different experiments. Unless the researcher is willing and able to look into the hidden layers and understand how each effects the weights of the next layer and ultimately the output, the workings of the net will continue to operate mysteriously.