Introduction

For this assignment, the goal is to build and train a Convolutional Neural Network to predict the steering angles and throttle to navigate a rover through the unknown terrain. After installing the Roversim application, the simulator was used to collect our training and testing dataset. Two different training and testing dataset was collected separately to perform task 1 and task 4. The dataset for task 1 is more nominal navigation compared to the dataset for task 4 because the rover was moved close to the cliff and obstacles in task 4.

Part 1
Data Collection

After finishing the recorded training ride stated above, the captured images and rover system parameters were saved. For example, in Julia's dataset, two rides were recorded, uploaded, and used as the training dataset and the test dataset, separately. There are 1545 images in the training dataset and 855 images in the test dataset. Because of hardware limitations, these images were resized to 180x180 to reduce the memory usage in the training step. In both datasets, the rover was driven in a natural operation manner. Each team member runs the simulator individually to collect training and test dataset. There are 4 training and testing data from each team member. This approach was used to ensure that there is enough training data and testing data to build and train a convolutional neural network.

Part 2
CNN Regression Network (Julia)

The CNN Regression network first does feature collection with two convolutional layers and then uses max_pooling and dropout layers to merge features. The network structure referenced a digit recognition network. The last layer of the network was modified, changing the activation function from softmax to relu and changing the output neural from ten to three. Early stop was applied to avoid overfitting.
(https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 178, 178, 32)      320

 conv2d_1 (Conv2D)           (None, 176, 176, 64)      18496

 max_pooling2d (MaxPooling2D  (None, 88, 88, 64)       0
 )

 dropout (Dropout)           (None, 88, 88, 64)        0

 flatten (Flatten)           (None, 495616)            0

 dense (Dense)               (None, 128)               63438976

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 63,458,179
Trainable params: 63,458,179
Non-trainable params: 0
```

2.3 Performance and Evaluation Metrics

The result was tested on the test dataset and evaluated with absolute error.

$$Absolute\ error\ =\ \frac{|y_{predict} - y_{grround\ truth}|}{y_{ground\ truth}}*100\%$$

Yaw absolute error=5.062837107211165%
Steer_angle = 16.608296481347717%
Throttle=14.26884035671493%

Discussion (Julia)
In this project, Yaw, Steering angle, and Throttle data are predicted. Yaw prediction has a relatively high accuracy of 95%. The training for Yaw is sufficient since Yaw data varies a lot during the training ride. The predictions of Steer Angle and Throttle have relatively high error rates of 16.6% and 14.2%, respectively. Because the road on the map is a Y shape, and the rover drives a straightway most of the time and only needs to turn in the middle point and three endpoints.

Discussion (Julia)

In this project, Yaw, Steering angle, and Throttle data are predicted. Yaw prediction has a relatively high accuracy of 95%. The training for Yaw is sufficient since Yaw data varies a lot during the training ride. The predictions of Steer Angle and Throttle have relatively high error rates of 16.6% and 14.2%, respectively. Because the road on the map is a Y shape, and the rover drives training, the Steer Angle is highly unbalanced, most of the time the steering angle is zero. The Throttle prediction also has a similar issue. In the training ride, I only accelerate the rover once and it slides by itself all the way until the turning point or the endpoints. If we have a more complex map, the predictions of throttle and steering angle could be improved.

Second, I understand that theoretically, if I use the RELU activation function in the final layer, the model can only output positive values, and the result should not be good. I should use the linear activation function in the final layer and let it be able to produce both positive and negative values. However, surprisingly, the result of the RELU activation function is quite good as reported. On the contrary, if I use linear activation function, the test error was really high(6 times) higher than the mean of the test values for both Throttle and Steer Angle. I think the reason why this issue happens is because the original network is fine-tuned. If I change the structure of the layers, the result may be seriously jeopardized.
For further information, please refer to the Julia folder, and Copy of hw3.ipynb under the Julia folder.

Part 3
Discretized Convolutional Neural Network (Felipe)

Code can be found on Github at https://github.com/leguizamofelipe/me_592

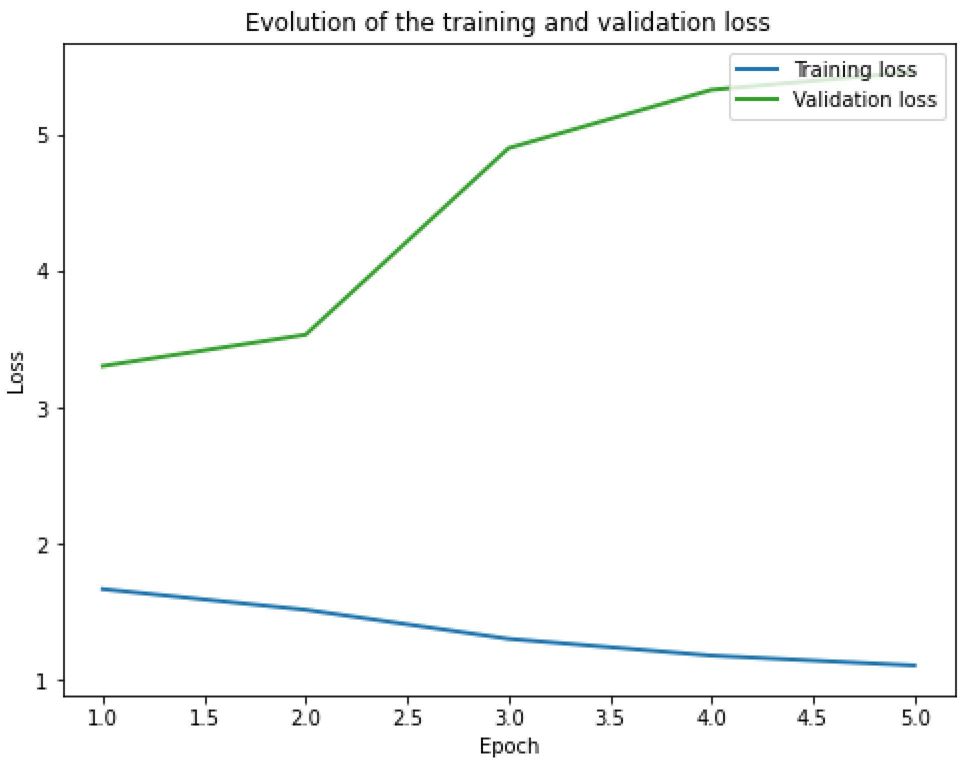
Reference File - HW3_Blended_Signals.ipynb
The assignment suggested discretizing the control signals into bins, so this could be done by looking at the sign of each signal. Since both throttle and steer angle could each be positive, negative, or zero (3 states), this allowed for 3x3 = 9 bins for classification. These were the bins that each image could fall under:

```
Steer/Throttle
0: -/-
1: -/0
2: -/+
3: 0/-
4: 0/0
5: 0/+
6: +/-
7: +/0
8: +/+
'''
```

Similar to the network used in class, this network took images as inputs and predicted the bins that each image would fall into. The network itself consisted of a convolutional layer, a MaxPool layer, and one hidden layer:

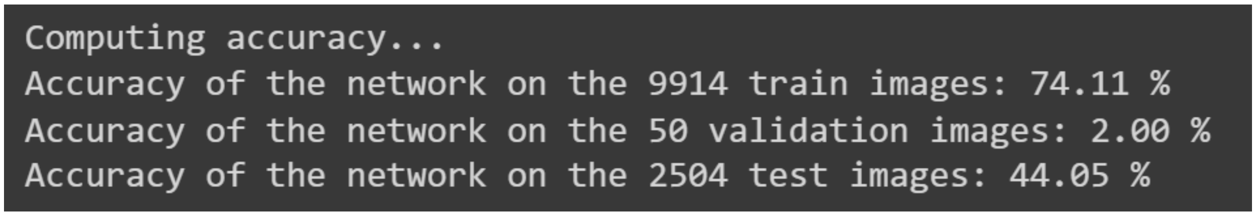
```
class SimpleConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(SimpleConvolutionalNetwork, self).__init__()
        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(18 * 16 * 16, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 16)
        self.fc5 = nn.Linear(16, 9)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = F.relu(self.fc5(x))
        return x
```

Dropout was implemented before the two linear transformations. The data consisted of a training set of 9914 images and a test set of 2504 images. The results of this architecture were mixed:



First, the validation loss was significantly higher than the training loss, and it was increasing during the training. This caused concern for overfitting, but adjusting the size of the network did not seem to help: layers and neurons could be taken away, but ultimately this discrepancy was not reduced. Furthermore, penalizing large weights (by setting a weight_decay in the arguments of the Adam optimizer) also did not help.

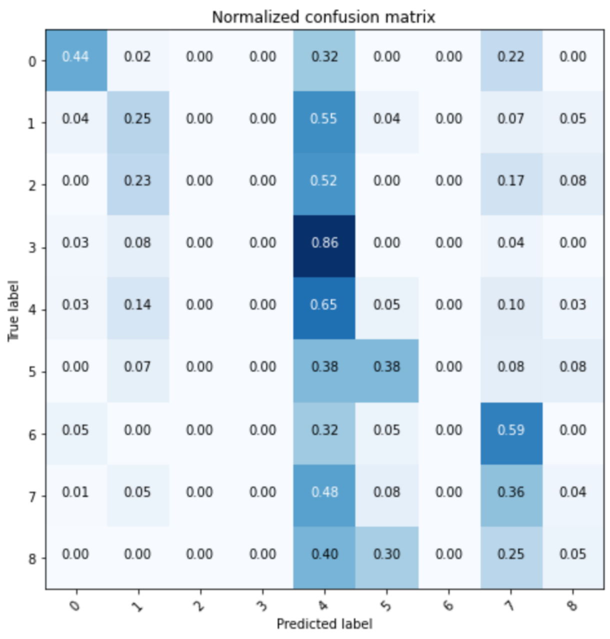
Despite this, the model did appear to learn: checking the test set showed an accuracy of 44%:.



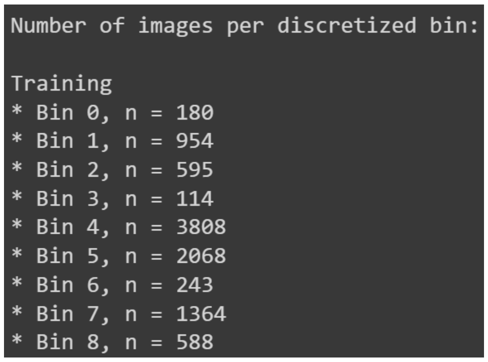
```
Computing accuracy...
Accuracy of the network on the 9914 train images: 74.11 %
Accuracy of the network on the 50 validation images: 2.00 %
Accuracy of the network on the 2504 test images: 44.05 %
```

It is not entirely clear why the validation loss and the test loss did not match up - it would appear that as random samples separate from the training set, they should present similar results.

The confusion matrix offers more information on the accuracy of the model:



Of note, a lot of incorrect predictions of the fourth state (neutral throttle, neutral direction) were made. This signals an inherent bias in the data: for much of the time, there is no throttle or direction command:



```
Number of images per discretized bin:

Training
* Bin 0, n = 180
* Bin 1, n = 954
* Bin 2, n = 595
* Bin 3, n = 114
* Bin 4, n = 3808
* Bin 5, n = 2068
* Bin 6, n = 243
* Bin 7, n = 1364
* Bin 8, n = 588
```

This is because the rover spends a fair amount of time simply driving around open spaces, and spends less time running into obstacles. Future training sets should include more instances of throttle and direction being commanded to avoid a wall or a rock.

Code can be found on Github at https://github.com/leguizamofelipe/me_592

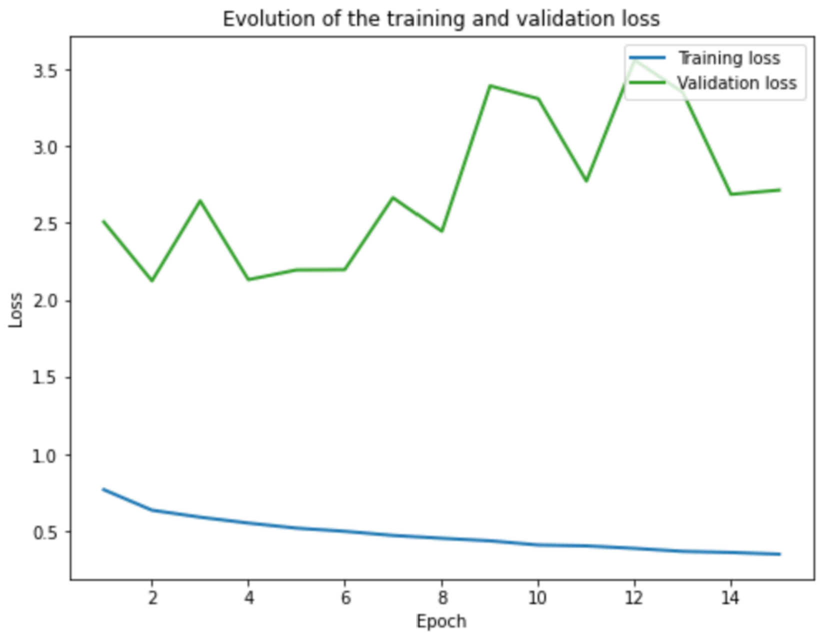
Reference File - HW3_Separated_Signals.ipynb
After discretizing into nine bins, it was decided to try splitting the signals. One model would classify the images by whether they were associated with a positive, negative or zero throttle, and another model would classify the images by positive, negative or zero steer angle. The neural network was slightly altered (by reducing hidden layers and using fewer neurons) to attempt and correct the overfitting seen in the previous attempt:

```
class SimpleConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(SimpleConvolutionalNetwork, self).__init__()

        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.dropout = nn.Dropout(0.2)
        self.fc1 = nn.Linear(18 * 16 * 16, 64)
        self.fc2 = nn.Linear(64, 9)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = self.dropout(x)
        x = x.view(-1, 18 * 16 * 16)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x
```

```
def createLossAndOptimizer(net, learning_rate=0.0005):
    # it combines softmax with negative log likelihood loss
    criterion = nn.CrossEntropyLoss()
    #optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
    optimizer = optim.Adam(net.parameters(), lr=learning_rate, weight_decay=1e-4)
    return criterion, optimizer
```

As shown above, a weight decay was added to the optimizer with the same motivation to avoid overfitting.

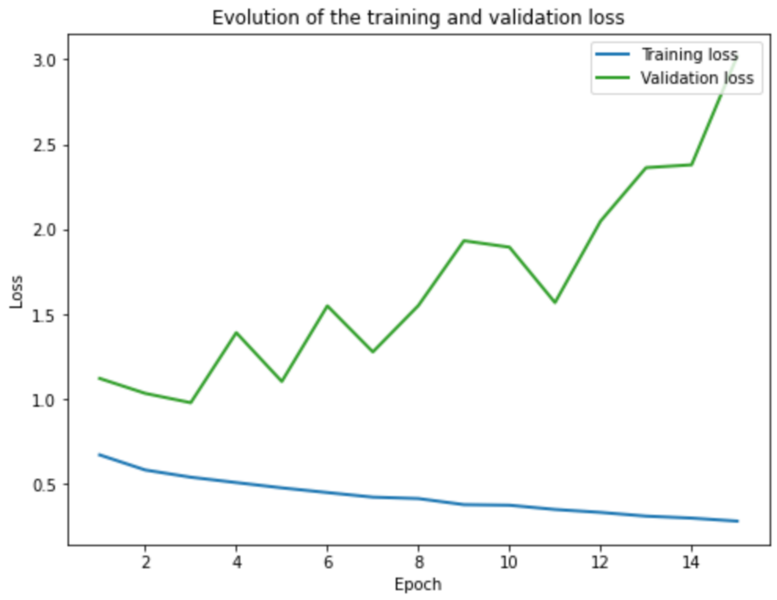
The results for classifying **steer angle** into positive, negative and neutral are below:



```
Computing accuracy...
Accuracy of the network on the 9914 train images: 80.84 %
Accuracy of the network on the 1000 validation images: 27.70 %
Accuracy of the network on the 2504 test images: 57.15 %
Class      Accuracy (%)
0            34.46
1            74.02
2            30.09
```

Once again, the validation loss was high, but the accuracy of the network on the test images was significantly higher. After the mitigation steps mentioned previously were taken, it is still uncertain why there is apparent overfitting. Once again, the accuracy was good for class 1 (neutral steer angle), but the other classes might have suffered from lack of data.

Finally, the results for classifying **throttle** tell a similar story:

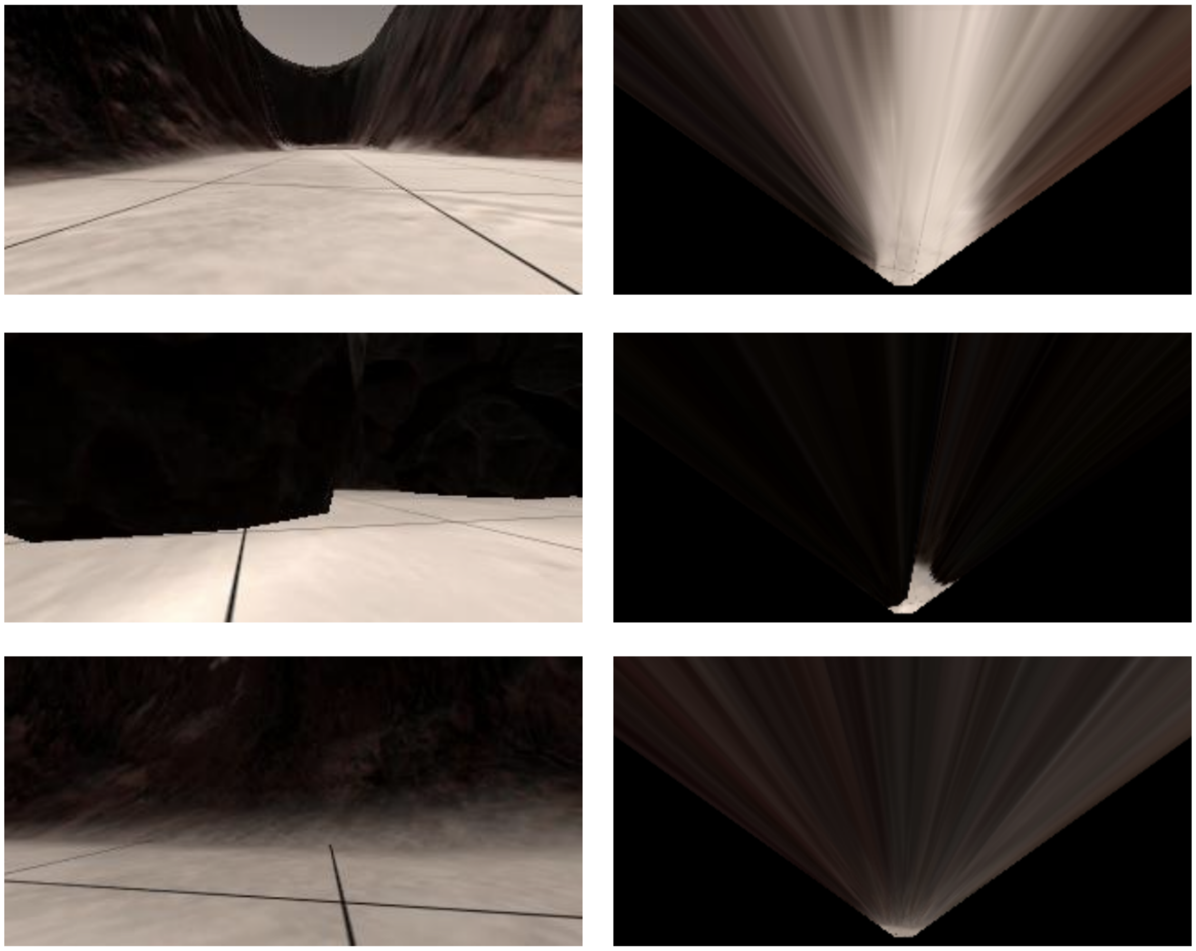


```
Computing accuracy...
Accuracy of the network on the 9914 train images: 81.93 %
Accuracy of the network on the 1000 validation images: 15.90 %
Accuracy of the network on the 2504 test images: 67.21 %
Class      Accuracy (%)
0            28.88
1            85.03
2            24.66
```

Here, the accuracy for classifying throttles on the test dataset was the best yet (67.21%), but this is muddied by the unbalanced accuracy per class (1 can be identified easily, 0 and 2 less so).

Ultimately, this training should be repeated with a more balanced data set. Given that the other remedies for overfitting have already been tried with little success, this might be the next logical step.

Part 4
CNN Regression Network
A different approach was used for this task. Instead of a classification model, a regression model was used. A different data preprocessing, perspective transform, was also implemented in this problem. The transform used in this task originates from the code provided in the repository on github. The data preprocessing part begins by turning on the grids in the sim and applying the perspective transform. The perspective transform seems to be really helpful in analyzing the data .One observation made was the larger the movable area within the image, the brighter the transform image. This means that the brighter the picture, the more movable space the rover has. This is very helpful for this task because when the rover is near a cliff, the transform image will be darker. The following figure shows the effect of the transformation: The left column shows before the transformation, and the right shows the after.

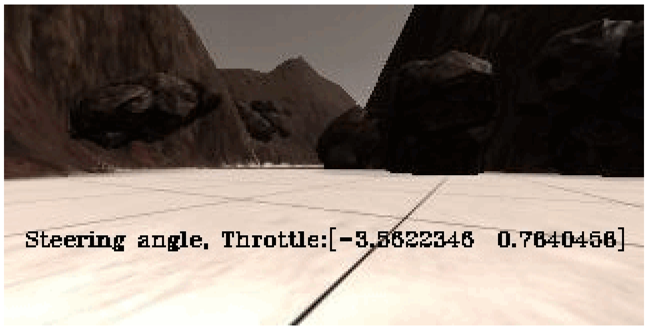


The architecture of the CNN is modified from the one provided during the hands-on session and available on canvas. The number of neurons and activation functions are modified to meet the needs of this problem. The following screenshot shows the architecture of the CNN:

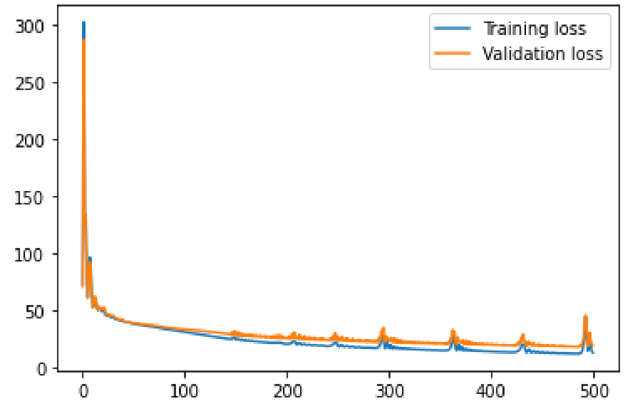
```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=8192, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=32, bias=True)
  (fc4): Linear(in_features=32, out_features=16, bias=True)
  (fc5): Linear(in_features=16, out_features=2, bias=True)
)
```

In this problem, we train the net on two different datasets, one normal behavior dataset with a total of around 8000 data points, and the another erratic dataset with around 4000 data points. To discuss the effect of the number of samples, another model was trained on a dataset with around 2000 randomly picked samples from the normal dataset. All models are then tested and evaluated on the same separately recorded testing data set.

Baseline



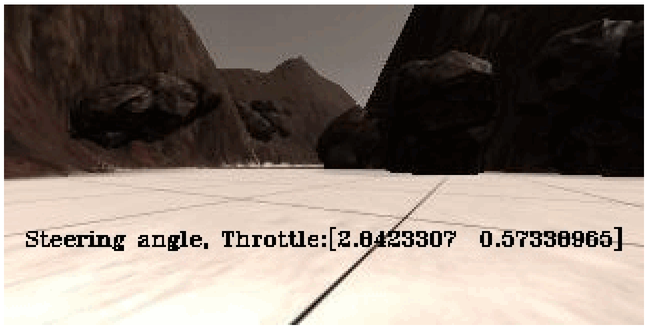
Steering angle, Throttle:[-3.5822346 0.7640456]

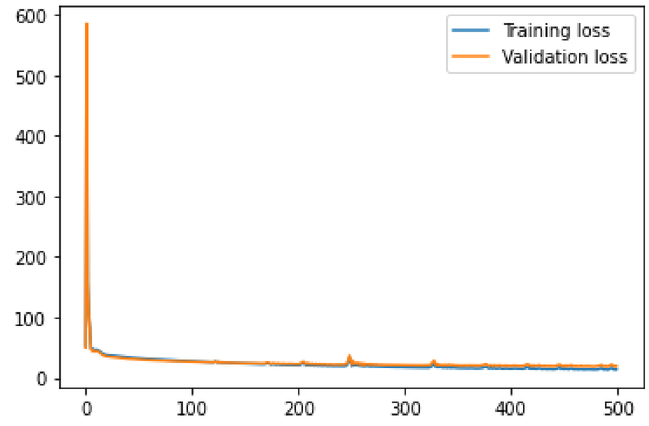


The baseline model is trained on the normal dataset, which has the most rational driving manner and the largest amount of data. The training shows slight overfitting as the epoch increases, so the model chosen for evaluation is from the 75th epoch, which is the point before the training and validation loss curves diverge. The reason behind the overfitting may be due to the sparsity of the training data points, since under normal circumstances, a rational driver would cruise on the straight and avoid unnecessary steering inputs. Such a behavior would lead to having many zeros in the dataset, which is definitely an issue for the network.

Still, the model seems to do a decent job on avoiding the cliffs. The gif above presents the results from the baseline model. (Note that the motion of the rover in the gif is independent of the predictions, only the numbers represent the predictions made by the model. The positive sterling angle stands for a left turn and a negative stands for a right turn). The model steers in the opposite direction when it is close to the cliff, and tries to follow the curvature of the path while at the center of the road.

Erratic





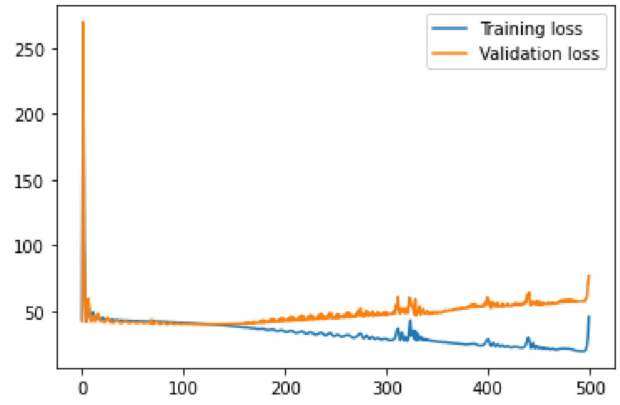
The erratic dataset includes much more erratic movements, including constantly steering left and right (even in the middle of the road), more dramatic throttle/brake application, and allowing at most two wheels to run over the edges of the cliff.

The training of this model ends up with a larger training loss when compared to the baseline, but surprisingly, it shows even less overfitting with fewer training samples. This may be due to the erratic driving style mitigating the sparsity problem we had on the normal dataset.

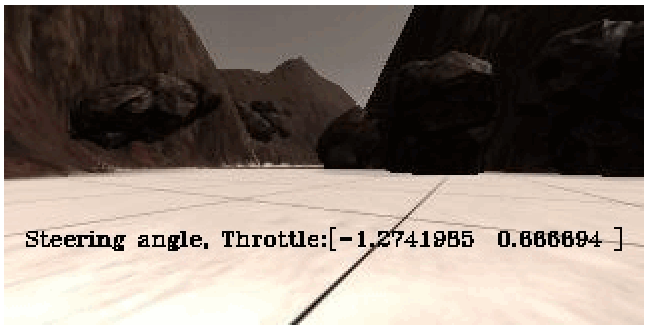
The model trained on the erratic data performed quite similar to the baseline model, but for some reason, it tends to steer right more than left when in the middle of the path. Nevertheless, it still manages to avoid the cliffs.

Less Samples

The less-sampled dataset is simply a dataset randomly sampled out of the normal samples. The net was trained on a quarter of the total amount of the normal dataset. Since overfitting is expected, we have chosen to save two models for comparison: one early-stopping and another over fitted model.

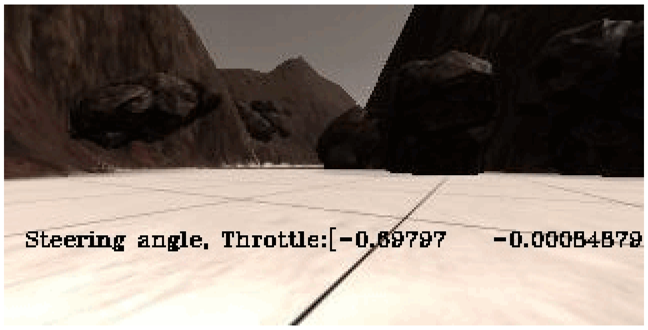


From the training loss plot, we can see overfitting obviously happening after epoch 110, and the average validation loss throughout the training period is also larger than the two previous models. The net is overfitting and memorizing the training data such that it becomes very sensitive to noise and hence performs badly on the testing dataset.



```
Steering angle, Throttle:[-1.2741985  0.666694 ]
```

The gif above presents the results of the early stopping model (epoch 100). The early stopping model still does a good job on avoiding obstacles, despite occasionally slightly turning into cliffs.



```
Steering angle, Throttle:[-0.69797    -0.00054579
```

The last gif shows the predictions of the overfitted model (epoch 499), we can see that the predictions will be completely off at some times, i.e, steering heavily into the cliff and applying brakes in the middle of the road with no obstacle to avoid.

For further information, please refer to the HW3_CNN_Regression_Problem4a.ipynb and HW3_CNN_Regression_Problem4b.ipynb under the HW3 folder.

Conclusion

Overall, different approaches were used for this project. The classification model and regression model were used to illustrate how different approaches can be used to build and train a convolutional neural network.