# End-To-End Steering Control Using Behavioral Cloning

**Mohamed Suhail, Mohitrajhu Lingan Kumarian**
**Raghav Hrishikeshan Mukundan, Sri Sanjeevini Devi Ganni**
University of Pennsylvania, PA, USA
{mohamedm,moji,raghavm,sganni}@seas.upenn.edu

## ABSTRACT

Autonomous Driving has been seen as a redemption from the chaotic on-road driving and the ruckus on the roads with the many advantages that it offers. In our project, we have focused on a small part of the self-driving car problem i.e. the steering control. Our motivation is to solve this problem by exactly mimicking the human-driven expert dataset with a learning approach known as Behavioral cloning. We have applied Computer Vision, Deep Learning, Non-Deep Learning, and Advanced Deep Learning models for this task and have compared the performance of all of them in this Image Regression problem. We have used a simplified simulator that can replicate the roads and the real-world environment to test our models. The second part of our project focuses on experimenting with Domain Adaptation i.e. to train the car on a new track and test how the car can work in the original track with few/no data of the test track (also termed as Zero-Shot Learning / Few-Shot Learning).

# 1 Introduction

Having a sturdy steering control is one of the crucial factors in Autonomous Driving. An ongoing open-ended problem statement is to make a robust steering control model for the self-driving car that is feasible for real-life scenarios with several companies making attempts to solve it. Earlier most approaches to solve this involved simple control, path planning, and vision techniques and recently with the advent of Deep Learning and Meta-Learning, there is a possibility of extracting features and driving well based on learning from a few training samples. Automobile tech giants like Tesla and Waymo have deployed their cars for testing and have shown promising results, yet not enough to convince the public about its prospective performance in real-world scenarios.

Steering control methods where the car is controlled following the carefully defined trajectories with sensory inputs from the environment may fail to adapt with real world conditions. Thereby, this results in increased crashes/accidents for autonomous vehicles. Between September 2016 and March 2018, Uber's autonomous vehicles were involved in 37 crashes and the majority involved the other vehicle striking the autonomous car, 25 of them being rear-end crashes.

A possible solution to this problem is to train cars with human-generated data and mimic it exactly for different scenarios with a method called Behavioral Cloning. Extensive training with this method helps the car to perform well in known environments. To investigate this, we first present and formalize an implementation that has the car running on the simulator without crashing/colliding. We have also simplified our problem statement with no external vehicles driving on the terrain. We then perform experiments that help in domain adaptation by testing it on a terrain that the model has little/no data.

## 1.1 Behavioral Cloning

Behavioral cloning, as per definition from the SpringerLink Encylopedia of Machine Learning, "is a method by which human sub-cognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that

gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce skilled behavior."

Behavioral Cloning is also the simplest Imitation Learning model where the neural network attempts to learn from a human demonstrator. In short, it can be considered as a supervised Imitation Learning problem. [1] This model can be used to make the car get accustomed to real-world challenges faced by a driver by supplying large amounts of training data from an expert driver. However, in our work, we have limited our usage of Behavioral Cloning to only steering control and speed control of the car (for the Multi-task model) only and in a simpler environment. It helps to make up for the shortcomings of the classical control techniques as it is modeled on the human-generated data.

The reason behind choosing Behavioral cloning over Reinforced Learning for this problem statement is because of the following advantages that Behavioral Cloning offers.

- Behavioral Cloning learns the optimal policy by mimicking the expert's behavior.
- By mimicking human-generated demonstrations, it can fare well in real-world scenarios.
- Simple and Sample Efficient as it gets the most out of every sample.
- Learns a set of demonstrations provided by the expert and hence there is no need to manually specify a reward function like in the case of Reinforcement Learning(RL).
- Behavioral Cloning is a Supervised Learning based technique instead of Reinforcement Learning (training on labeled data).
- It takes lesser training time when compared to RL.

The expert-driven dataset consists of labeled RGB images with steering angle, speed, throttle and brake force values for every instance of the car motion. We apply Behavioral Cloning using a CNN on this dataset to regress and reduce the loss between the predicted and ground truth steering angles. It is tested in real-time on the simulator when the car is in Autonomous mode. We tried using different architectures and pre-processing techniques to compare the results. We have also come up with a Non-Deep Learning Baseline using Support Vector Regression (SVR) to apply Behavioral Cloning and compare it with the DL baseline. We have implemented Advanced DL baselines like a Multi-Task Learning model with an LSTM that controls speed and steering simultaneously and also a Few-Shot Learning model to perform domain adaptation.

## 1.2 Domain Adaptation

This part of the project is an extension of the Behavioral cloning part. Domain adaptation is a scenario in which a model trained on a source distribution is tested on a different target distribution. It uses labeled data in the training domain and tries to solve new tasks in the test domain to transfer knowledge gained from training on labeled data when testing on unlabelled data.

In this project, we train on a particular dataset and test on data that the model has not seen or seen very little of. We try to adapt across two terrains namely: Jungle and Desert. This is done to see how a model trained only on a particular dataset learns features and performs in a completely new environment. We train on the Jungle terrain and test on the Desert terrain.

In a bid to make a model adapt across two domains, we have experimented with different Deep Learning architectures, image pre-processing and feature extracting techniques like HOG ( History of Gradient) transformations, Canny Edge Detection and Unsupervised Segmentation. We used Transfer learning (a pre-trained Resnet-34 model on Imagenet) and a model inspired from the Relational Network paper to implement few-shot learning for our advanced Deep Learning baselines for Domain Adaptation.[2] The Non-Deep Learning baseline using SVR was used for Domain adaptation as well.

# 2 Related Works

Autonomous Land Vehicle in a Neural Network(ALVINN)[3] is one of the earliest successful neural network based self-driving vehicle projects. The network in the model is simple and shallow, but it manages to do well on a simple road with few obstacles. The recent advancements in Deep Learning[4, 5] has led to the use of Convolutional Neural Networks for environment perception and steering angle prediction.

Nvidia is the first to adopt Convolutional Neural Network for End-to-End steering control in their model named the PilotNet[6]. In this system, they propose a model that uses three front-facing cameras and successfully demonstrated that the car can be controlled in an end to end manner. There are two methods to control steering angle from visual inputs: Behavior Reflex Approach and Mediated Perception Approach. In the Mediated Perception Approach[7], the visual input is first mapped to predefined parameters. Then, the Rule-Based methods generate control commands with the estimated parameters. Although these methods lead to smooth vehicle control, they work in very limited scenarios. In the Behavior Reflex Approach[8] the steering angle is directly predicted from the visual inputs. These models are less complex and can be more robust with enough training data. Furthermore, these models also have good generalization ability.

However, these models fail to perform well in complex environments. This is because these models only control the steering angle. The end-to-end autonomy of vehicles in complex environments requires a model that controls multiple vehicle control parameters simultaneously leveraging as many input data from the vehicle as possible in addition to visual inputs. Yang.Z *et.al.*[9] and Chowdhuri *et.al.*[10] propose Multi-Modal Multi-Task Learning Networks which can simultaneously control both speed and steering angle. These models are an extension of the Behavior reflex Approach with the Multi-Modal Multi-Task framework. Feedback speeds are used as an extra modality for steering angle and speed predictions.

In an ideal situation, a model that has learned a specific task should generalize to a new dataset collected for that specific task. But, experiments and research reveal that the performance of the model deteriorates rapidly on an input from a different distribution for the same task. This is mainly due to the inherent bias introduced in the training data during the data collection process. This phenomenon is known as dataset bias or domain shift. Simple models such as Kernelized SVRs[11] learn important features from the training data and can generalize across different domains. But, Complex models learn these biases in addition to the necessary information for that task.

Most models used for autonomous driving tasks these days are very complex. In the world of Autonomous Driving, it is very critical to have a model that can generalize to unseen scenarios/ to scenarios that have very few annotated training data. The model should be able to transfer knowledge across different domains. This process is called Domain Adaptation.

The main aim of domain adaptation is to de-bias the models and pick the right set of parameters so that the model learns only to pick the task-specific information from the domain and ignores the inherent biases. In recent years, many researchers have worked on domain adaptation for perception models. However, most of these works focus on image classification or semantic segmentation models. Tzeng*et.al.*[12] propose an Adversarial adaptation framework for classification task on MNIST dataset. D.Sahoo*et.al.*[13], F.Sung*et.al.*[2] propose a few shot learning paradigm for image classification tasks.

In this work, we develop a Non-DeepLearning baseline model using Kernelized SVR for steering angle prediction that incorporates HOG-transformer as a preprocessing step. We also develop the Behavioral Reflex CNN model and a Multi-Modal Multi-Task model with necessary preprocessing as the deep learning baseline. Further, we check the generalization ability of these models across different terrains. We also experiment with the Unsupervised image segmentation[14] and canny edge detection[15] techniques to aid the models to learn the necessary information ignoring the bias. We also try a transfer learning model to see if it can generalize across different terrains. Inspired by previous works on Meta-Learning for domain adaptation, we attempt to develop a model incorporating a few-shot learning paradigm to obtain a model that can generalize for steering angle prediction across different domains as our advanced Deep Learning baseline.

# 3 Simulator and Dataset

We used the open source Udacity Self-Driving simulator to test our models. For training we used the Kaggle dataset from [16]. The simulator has two tracks- Plains(Dessert) track and the Hilly(Jungle) track. We test the model using the autonomous mode in the simulator. Throttle(acceleration/deceleration) and steering angle are used to control the vehicle in the simulation. We have explained more about the simulator and dataset in detail in the Appendix section.

# 4 Methods

In this section, we outline the various model baselines and techniques that we have implemented for end-to-end steering control and domain adaptation. We start with a Non-Deep Learning Baseline developed using Support Vector Regression(SVR). This model was trained to predict steering angle for Track1 (Desert) taking the image sequence as the input. Following this, we developed a Deep Learning baseline using Behavioral Reflex CNN model and trained it on track1. To study the effect of domain shift on these models, the aforementioned models were later trained on a new dataset Track2 (Jungle) to see how well the models could run on Track 1 (Desert). We experimented with some image processing techniques such as Canny Edge Detection and Unsupervised Image segmentation to see if these techniques help us improve the generalization ability of the models. We also iplemented a Transfer Learing model to test its capacity to generalize across domains.

Building on this, we tried two Advanced Deep Learning Baselines and experimented with these approaches for Domain Adaptation to see how good they could perform. First, a Multi-Modal Multi-Task Learning model that was capable of simultaneously predicting the steering angle and speed was applied.

Finally, a model inspired by the idea from the paper on Relation Network for Classification[2] was developed to suit our needs and to see whether it could perform Domain Adaptation. In the initial implementation of the model, we use a few samples from Track1 along with the Track2 dataset for the training process, adhering to the usual Few-Shot Learning approach. We extended this by training on a support dataset apart from the Track2 dataset to help in performing domain adaptation.

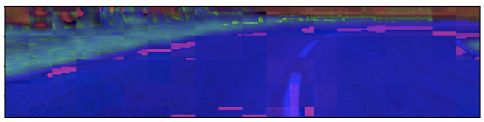
## 4.1 Pre-Processing

We perform a set of operations to make our model generalize and prevent overfitting. All the images are cropped at the top and bottom to exclude the bonnet and a part of the sky. Cropping the image helps remove the irrelevant features. We further implement functions for flipping the images, translating the image on both the x and y-axis. Flipping the image by 180 degrees helps us have a dataset with equal left and right turns. The dataset we have has predominantly left turns, so this step helps in fixing the biases in training data. Translating the images shifts the image so that position of the vehicle is generalized.

Different scenarios have different brightness and shadows depending on background information like trees, mountains,etc. We also create functions that add shadows to the images in different random places. For this, a few coordinates are chosen and then we mask those areas with zeros. The image is converted to HLS format to adjust saturation and then converted back to RGB. The brightness is also adjusted based on random values generated. All these functions are implemented with the help of the CV2 library.

The random images are flipped and translated. Based on chance, shadows and brightness adjustments are also done. Finally, all the images are converted to HSV(Hue-Saturation-Value) format.

*Original Image



*Pre Processed Image with Random shadows, translations, shadows and brightness adjustments

Pre-Processing the Image

## 4.2 Data Augmentation

There are three cameras on the vehicle in the simulator, the left camera(on the left side of the bonnet), the right camera(on the right side of the bonnet) and the center camera(in the middle of the bonnet). Hence, we get three images from the simulator for a particular scenario.

To increase the training data, we train the model on all three images. We adjust the steering angle on the images from the left and right cameras to make them consistent with the center images. Further, we also flip the center image for additional data augmentation. This helps in balancing the left and right turns in the dataset as the dataset has more left turns compared to right turns.

## 4.3 Non Deep Learning Baseline - Support Vector Regression

We chose Support Vector Regression (SVR) as our Non-Deep Learning Baseline for this image regression problem. SVMs (Support Vector Machines) which are essentially used for classification problems, were extended to SVRs in 1996 by Vapnik*et.al.*[11]. Similar to SVMs, the SVR model depends only on a subset of the training data, as the cost function for building the model ignores any training data does not care about training points that lie beyond the margin.

SVRs are trained by solving this: [17]

$$\begin{array}{ll} argmin_w & \{\frac{1}{2}||w||^2\}. \\ \text{such that} & |y_i - wx_i - b| \leq \varepsilon. \end{array}$$

where $x_i$ is a training sample with target value $y_i$. The inner product plus intercept (b) is the prediction for that sample, and $\varepsilon$ is a free parameter that serves as a threshold ie all predictions have to be within an $\varepsilon$ range of the true predictions.

After obtaining our dataset, we wanted to lower the complexity of the images while maintaining the variances of the images as much as possible before fitting the SVR model over them to produce better training. We used the HOG (History of Gradient) transformation from the scikit image package on every image before appending them to the training list. HOG is a feature descriptor that converts an image to a feature vector of numpy array type. In HOG, an image is divided into blocks and the magnitude of the gradient in a given number of directions is calculated as feature vectors. Every image was gray-scaled before applying HOG using the 'rgbtogray' function from the scikit-image package.

The SVR model was applied over the processed training data using the SVR function from scikit learn package. The regularization parameter 'C' was set to 1 by default. The kernel function was also left to the default rbf(radial basis function). The epsilon or the threshold for true predictions was set to 0.2. The pickled representation of the model is saved to a file using 'pickle.dump' function from the Pickle package.

The images rendered by the simulator during the telemetry process are the testing images. These incoming images are transformed using History of Gradients (HOG) again and the pickled

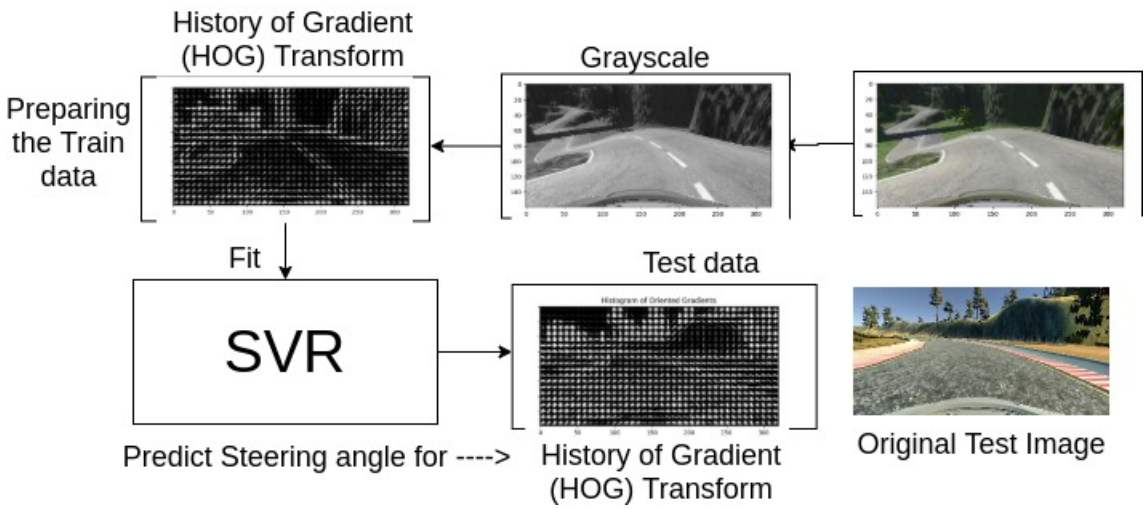



Figure 1: SVR trained on Jungle Track

SVR model is loaded over them to predict the steering angle values for this testing data.

## 4.4 DEEP LEARNING BASELINE

### 4.4.1 Train/Test on Track-1

We perform Single-Task Learning to calculate the steering angle of the car. There is only one model that takes in the image as input and returns the steering angle at the end of the network.
From the following figure, we can see that Single-Task Learning represents a one-one mapping between the training data and the model. This means that for every training data, there can be only one model that is using it.

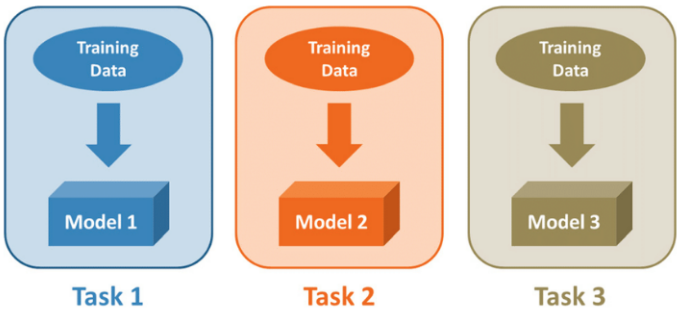



Figure 2: Single Task Learning

Advantages of using Single-Task Learning:

- Easy Representation
- Trains fast
- Simple and Sample Efficient (it gets the most out of every sample).
- Works well with good training data , pre-processing and network architecture

Disadvantages of using Single-Task Learning:

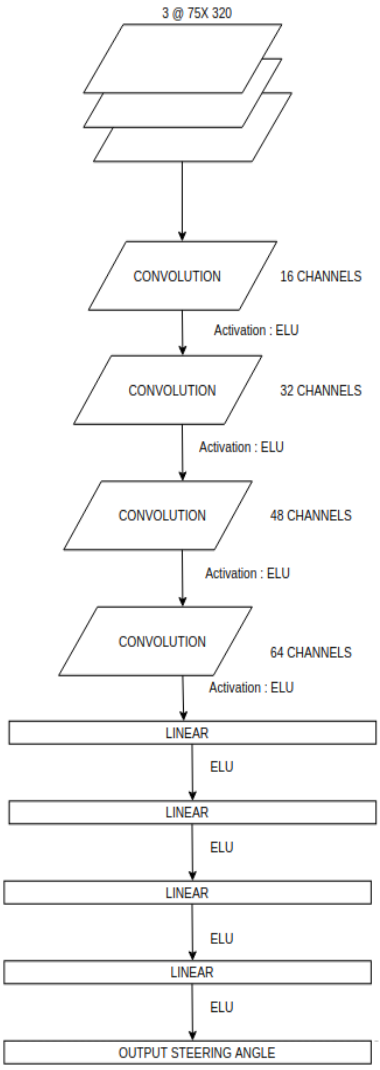



Figure 3: Architecture

- Highly dependent on the training data
- Ability to generalize across terrains is less compared to simple models like SVRs
- Only controls Steering angle. Thus performance is only limited to environments where speed control isn't crucial.

This Single-Task Learning model runs perfectly without colliding on the track that it is trained on.

**ARCHITECTURE**

Figure 4 shows the architecture that has been used for the CNN. It uses the Desert terrain as the training data and runs successfully on the terrain without crashing. We observed that the model got better with the number of epochs. This is set as the Deep Learning Baseline for track completion. We repeatedly experimented with multiple architectures and finally arrived at the above network that made the car perform well without colliding with the environment. We initially used ReLu as activation function unit between the layers but were able to find a significant increase in performance when replaced with ELU.

**HYPERPARAMETER TUNING**

We implemented the Adam optimizer and the learning rate was adjusted repeatedly until the optimal learning rate was found. We tried adding Dropouts and Batch normalisation in between the layers but

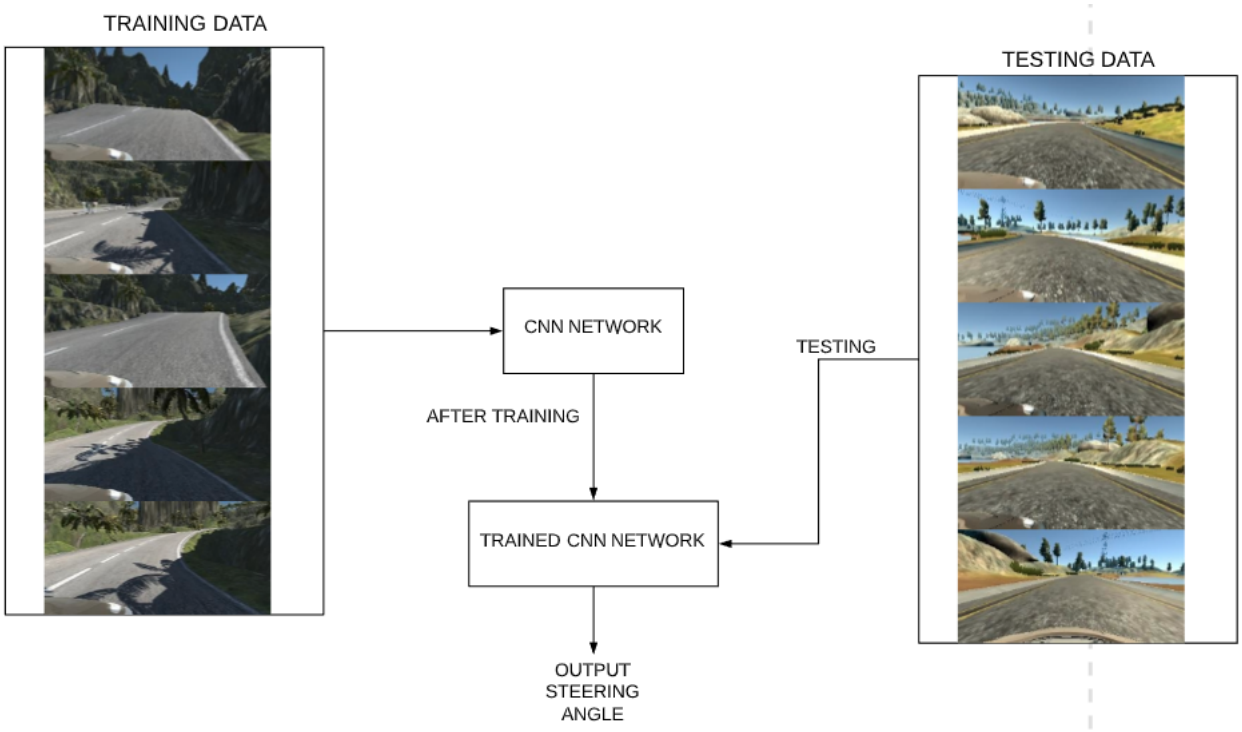



Figure 4: Zero-Shot Learning

it didn't improve the overall performance of the model. There was one sharp right turn in the track which required some tuning. We had to add few more images of that particular point in the dataset and randomly translated the images during pre-processing to provide more information about the turn to the model.
These are the final hyperparameter choices:

- Loss = Mean Square Error Loss
- Batch Size = 32
- Learning Rate = 0.00005
- Optimizer = Adam

### 4.4.2 Domain Adaptation

In this section, we try to make the car to steer without crashing in the terrain that it has no information about. This is called **Zero-Shot Learning.** It is when a learner observes samples from classes that were not observed during training, and tries to predict the values. The Track 2 (Jungle) is trained and the testing terrain is Track 1 (Desert). The aim is to make the car to run successfully on Track 1. We observed that the current architecture does not work for generalization and implemented a new architecture for the CNN.

There was additional data pre-processing that was needed. Zero-Shot Learning relies on the training data where it tries to extract as many features and use it while seeing unseen data during testing. While doing pre-processing, we performed a random flip to equalize the left and right turns. There was also translation performed randomly to make the model not only look at the center image. We also generate a shadow randomly to match if any is created by objects like the tree. We also change the brightness randomly to equalize brightness over the entire terrain.

**ARCHITECTURE**
The architecture that is shown below in Figure 6 helps in making the car to steer well in the Desert

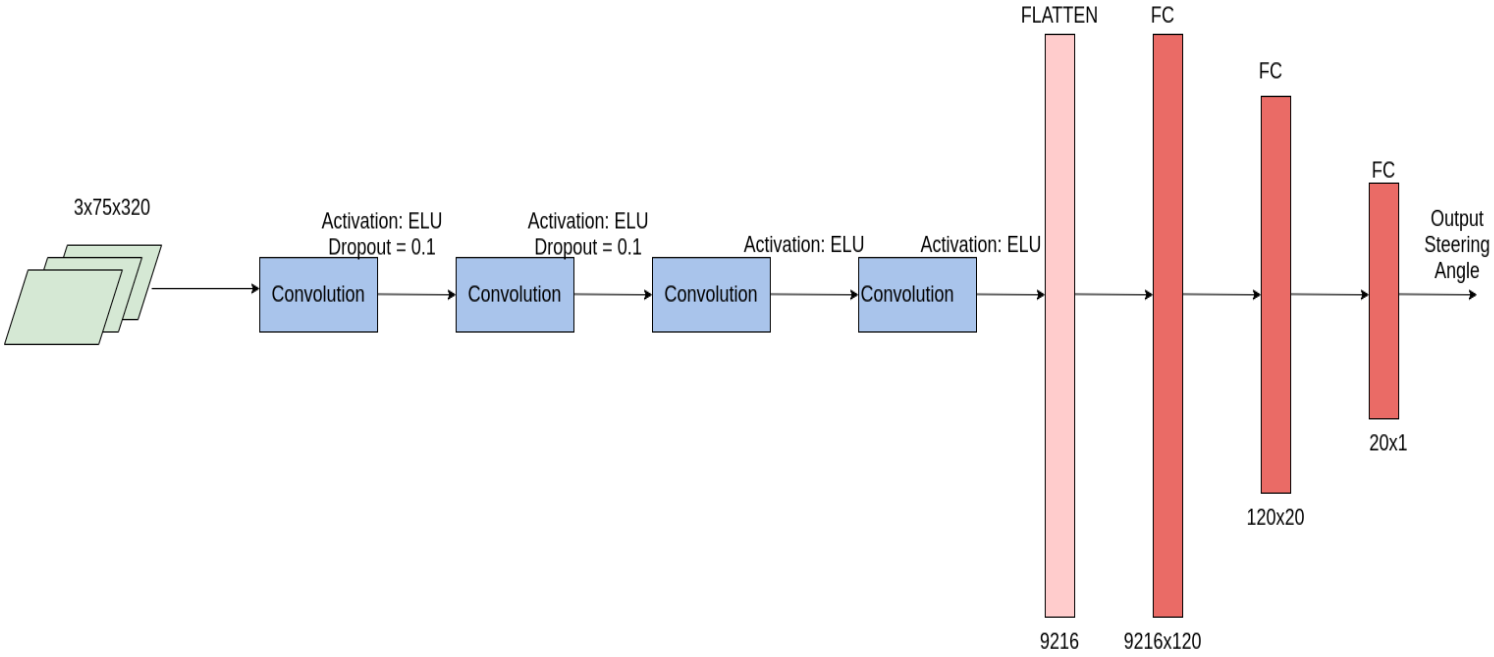



Figure 5: Architecture used for Domain Adaptation

terrain after training in the Track2 dataset. It was observed that the model performed well with less number of training samples. Therefore, the dataset was reduced to 4000 samples. With little data of the Jungle terrain given, the car was able to complete the Desert terrain without crashing/colliding. We were able to conclude that when the neural network was able to learn numerous unnecessary features initially and couldn't perform well and thus we reduced the training data.

**HYPERPARAMETER TUNING**
The learning rate was adjusted repeatedly until the optimal learning rate was found. The Adam optimizer was chosen as the final optimizer. Weight decay helped in preventing the weights from becoming big and was part of the regularization. Few pre-processing
These are the final hyperparameter choices:

- Loss = Mean Square Error Loss
- Batch Size = 32
- Learning Rate = 1e-4
- Optimizer = Adam
- Weight Decay : 1e-5

**ATTEMPTS TAKEN TO IMPROVE DRIVING :**

We made a few unsuccessful attempts at improving domain adaptation that either didn't perform well or are computationally expensive to produce descriptive results. We initially started with the Canny Edge detection, a famous multi-stage edge detection technique was used for lane detection and we were able to see it fail for curved roads. Also, one of our attempts was to apply unsupervised image segmentation with a CNN to distinguish the road from the environment while pre-processing.

However, pre-processing takes two minutes for every image and hence it is not feasible.

Siamese Networks are usually synonymous to One Shot Learning. It passes the two images as input and calculates similarity between the images. It uses a loss function called the contrastive loss that takes in the euclidean distance between the two images. We did not use this for the model as this couldn't learn enough features from all the three cameras. We also tried a transfer learning model pre-trained on ImageNet with the last linear layer replaced to predict the steering angle. It fails to detect the minute differences in orientation of the car and the road, thereby causing the car to crash in the few seconds.

More details and analysis of these models are included in the Appendix section.

## 4.5 ADVANCED DEEP LEARNING BASELINES

### 4.5.1 Multi Task Learning for Speed and Steering Control

Until now, we only learn to control the steering of the vehicle. For controlling the speed, we define the minimum and maximum speed limits and calculate the throttle using the formula $throttle = 1.0 - steering\_angle * *2 - (speed/speed\_limit) * *2$. Where $speed\_limit$ is the minimum defined speed if the current speed is greater than maximum speed else the $speed\_limit$ is the maximum defined speed. This arrangement works but is not our end goal- to achieve end-to-end of the vehicle. Thus, we aim to train a model that predicts the throttle along with the steering angle.

To realize this we can develop two separate Single Task Learning models, one for predicting the throttle and another to predict the steering angle. Although we can achieve acceptable performance this way, we ignore information that can potentially help us generalize better and improve performance. Specifically, this information comes jointly from the signals of these two related tasks(control over throttle and steering angle) variables in the training set. By sharing representations between the two related tasks, we enable our model to generalize better on the original tasks. This is why we use Multi-Task Learning.

In [9], the authors propose a multi-modal multi-task network for vehicle control. They use a Convolution network to interpret the input image and further use fully connected layers to predict the steering angle. For, the speed control they use an LSTM with the speed sequence as input. They concatenate the output of the LSTM with one of the linear layers from the steering control part to reuse the information from the image. Using this paper as inspiration, we implement multi-task learning over the Single task learning architecture implemented in the deep learning baseline section.

**ARCHITECTURE:** We slightly change the architecture from the previous section to incorporate the throttle architecture. For the loss part, we added the weighted losses of both the steering and throttle outputs. We tried to find the best ratio for both losses. We trained this model on the Desert dataset and tested it on the Desert terrain. We tried to generalize it to domain adaptation but were unable to do so.

**HYPERPARAMETER TUNING** Similar to previous section we used Adam Optimizer with weight decay. We also adjusted the learning rate to find the optimal learning rate. The final hyperparameters are:

- Loss = Mean Square Error Loss
- Loss Weights = 0.99 * Loss_1 + 0.01 * Loss_2 (where Loss_1 is the loss of the steering angle and Loss_2 is the loss of the throttle)
- Batch Size = 32
- Learning Rate = 2e-6

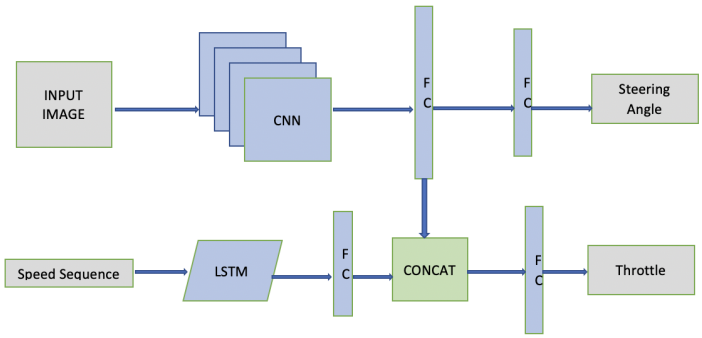



Figure 6: Multi Task Learning Architecture

- Optimizer = Adam
- Weight Decay : 1e-4

### 4.5.2 Few-Shot Learning

Few-shot learning is when we feed a learning model with a very small amount of training data and perform testing. We give three classes of images(images depicting left, right and straight movements) of the Desert terrain to the CNN along with the input image from the Jungle terrain. This would help the model to learn features from the three types of images and find it easier during the testing process.

**ARCHITECTURE**

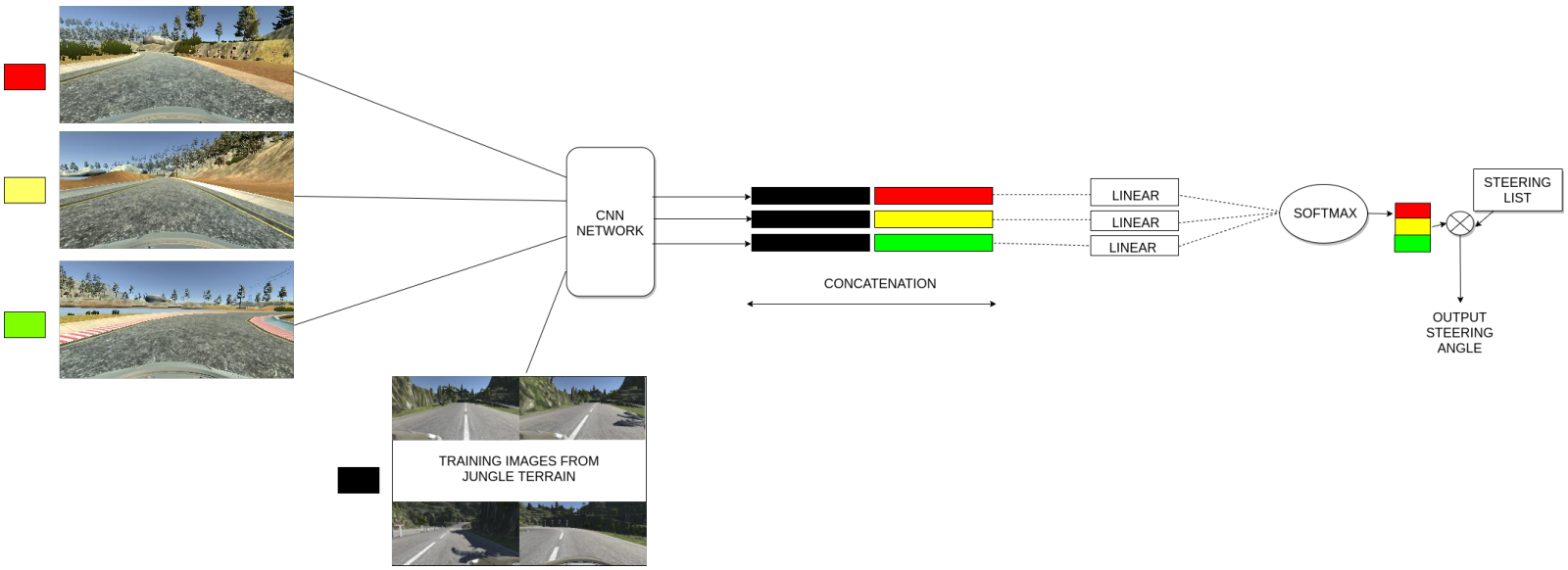



Figure 7: Few Shot Learning

In [18], Sung et.al. propose a relation network for few shot learning. They implement a classification network. We take inspiration from their work and try to implement a regression network. Our way of implementing the regression network for few shot learning is novel. The images from the Jungle terrain and the Desert are different. We want the model to find the similarities between the two terrains. To achieve this we want the CNN to convert the images into embeddings. Thus we use these embeddings to find the similarity of the image with the fixed images. Using these embeddings we should be able to calculate the similarity vector of the input image with the fixed images. Now using this similarity vector and the steering angles of the fixed images we will be able to predict the

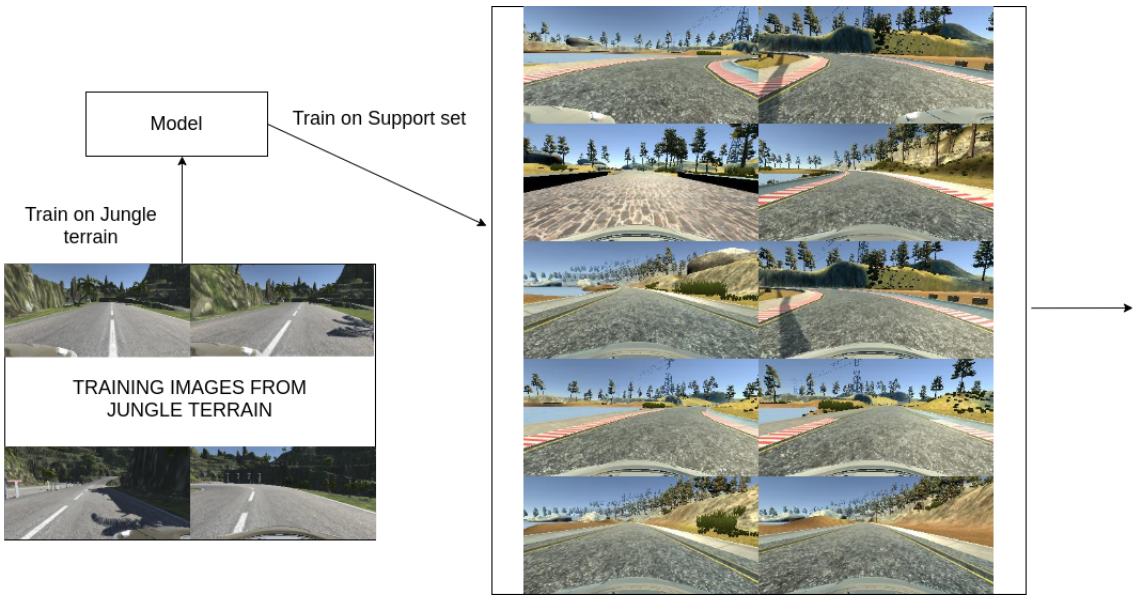



Figure 8: Training with Few Shot

steering angle of the input image.

The embedding size is set to be 128. That is, we get four vectors of 128 each from the CNN network(one for the input image and the other three feature vectors of the fixed images). The three feature vectors are concatenated with the input image to give three vectors of size equal to 256. This is then passed into another linear network that gives an output vector of size 1. We concatenate these vectors to get the similarity vector. The similarity vector has to be a probability vector(elements sum to one). Hence, we pass this vector through a softmax layer. This vector says how similar the input image is to each fixed image. To find the resulting steering angle we decided to multiply the vector with the steering angle list. The Steering angle list contains the steering angle of the three images from the Desert terrain that we insert into the CNN network. The result is the output steering angle.

**HYPERPARAMETER TUNING:**
The loss was set to MSE loss as it is a regression problem. The learning rate was changed from 0.1 until 0.00001 with different values. Finally, a learning rate of 0.00001 was chosen after it gave the best results. The weight decay was chosen as we did not want the weights to become big.

- Loss = Mean Square Error Loss
- Batch Size = 32
- Learning Rate = 1e-5
- Optimizer = Adam
- Weight Decay : 1e-3

**TRAINING PROCESS:**
The training data is from the Jungle terrain. In an extension to the existing implementation, We use a support set that we train on apart from the training set. The support set is a set of 8 images from Desert terrain. After every batch iteration we train the model on the support set. This is to bolster the performance during testing and the make the model more domain adaptable. We train the model for 15 epochs with hyperparameter tuning. We understand that we need more computation resources to be able to get the few shot to work properly.

# 5 Analysis

## 5.1 Evaluation Metrics

We considered the following metrics for evaluating the performance:

- Loss: The loss function of the model, mean square error between the predicted outcome and the true value.
- Difference between the Ground Steering Angle and Predicted Steering angles: This plot helps in identifying the range of values predicted by the model and analyse any trends in the predicted angles.
- Track Completion: The percentage of the track successfully covered by the model. Using this, we can see how well the model works on steep turns and different environments. The track in the Desert terrain has a bridge and turns with varying steepness. We introduced this metric as loss is not a sufficient metric to evaluate the model.

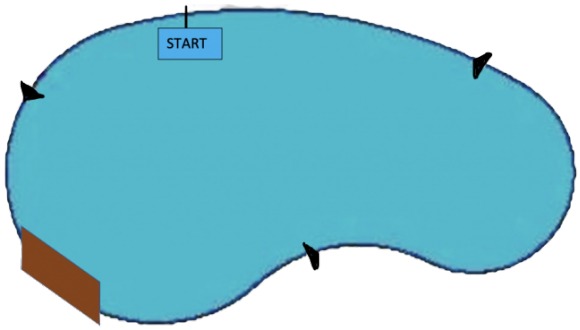



Figure 9: Track Shape

## 5.2 Support Vector regression

Our non-deep learning baseline was trained on the Jungle track. It performs well on both the Jungle track and Desert track. It can cover both the terrains without crashing anywhere. The HOG transform helped the model to generalize the features on both the tracks, thus enabling domain transfer.

From the steering angle graph and the training loss graph, we observe that the predicted value and the ground truth are very similar. The predicted steering angle fluctuates a lot causing the vehicle to wobble a lot while running on the tracks. The SVR model is unable to give consistent steering angles for a smooth ride.

## 5.3 Deep Learning Baseline:

We observe the DL baseline completes the Desert terrain when trained on it. The following graphs show the losses while performing training, validation and also compares the steering angle to the ground truth. It performs well when there are a few samples provided.

The model performs very well when trained on the Jungle track. It is able to complete the track in the Desert terrain. From the Steering Angle graph, we can see that the predicted steering angles do not make any sharp movements and have consistent steering movements. In the Deep Learning baseline, the vehicle does not wobble. This makes this model better compared to the non-Deep learning baseline.

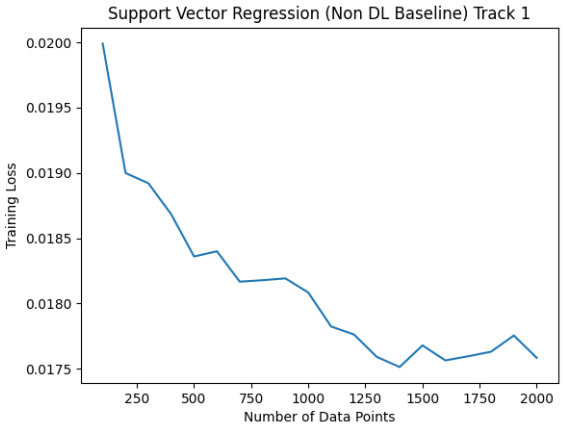



Figure 10: Support Vector regression - Training Loss

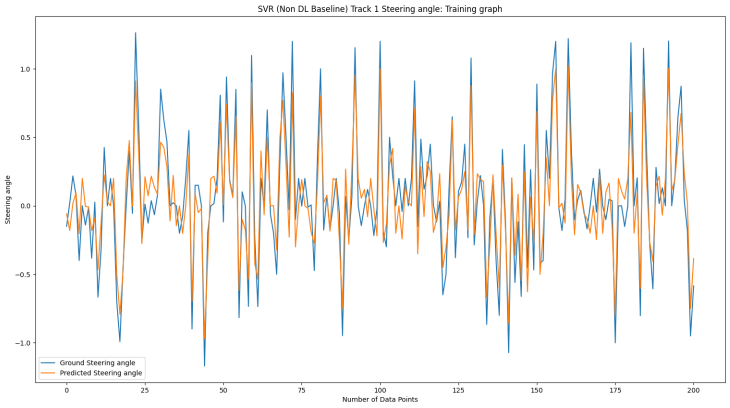



Figure 11: Support Vector regression - Steering Angle Graph

## 5.4 Advanced Deep Learning Baseline:

### 5.4.1 Multi Task Learning

For multi task learning, we train the model on the Desert track and test it on the Desert track. The model is able to complete 50% of the track and crashes at the steep right turn. The graphs for the training, validation loss and steering angle are shown below.

Our main objective of using Multi-Task learning is to control both the speed and steering angle. Our model was able to regulate speed pretty well. From the steering angle graph, we can observe that the predicted steering angle is pretty much constant with little fluctuations. This could be the reason why the model fails on the same domain after completing 50% of the track. The loss is also higher in general compared to other experiments as the loss function is a combination of both steering loss and throttle. The objective of this model was to check whether the model was able to control steering angle and speed simultaneously for a significant period of time and we were able to visualize the results in the simulator.

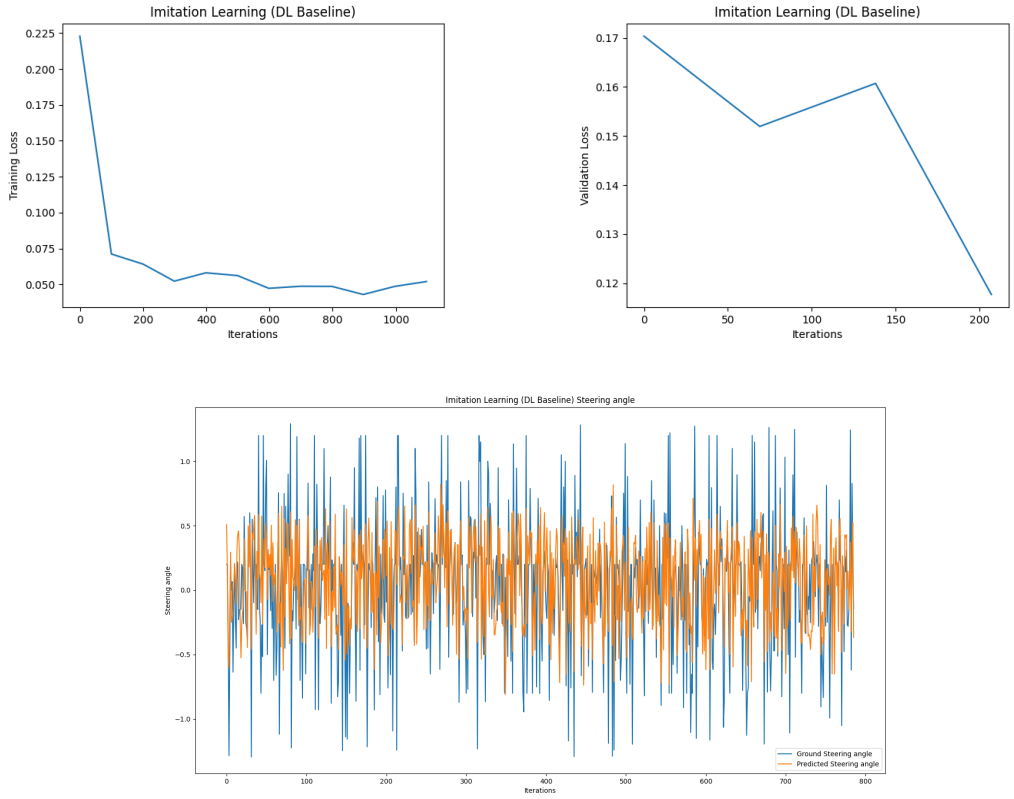



Figure 12: Training Loss, Validation Loss and Steering Angle- Deep Learning Baseline

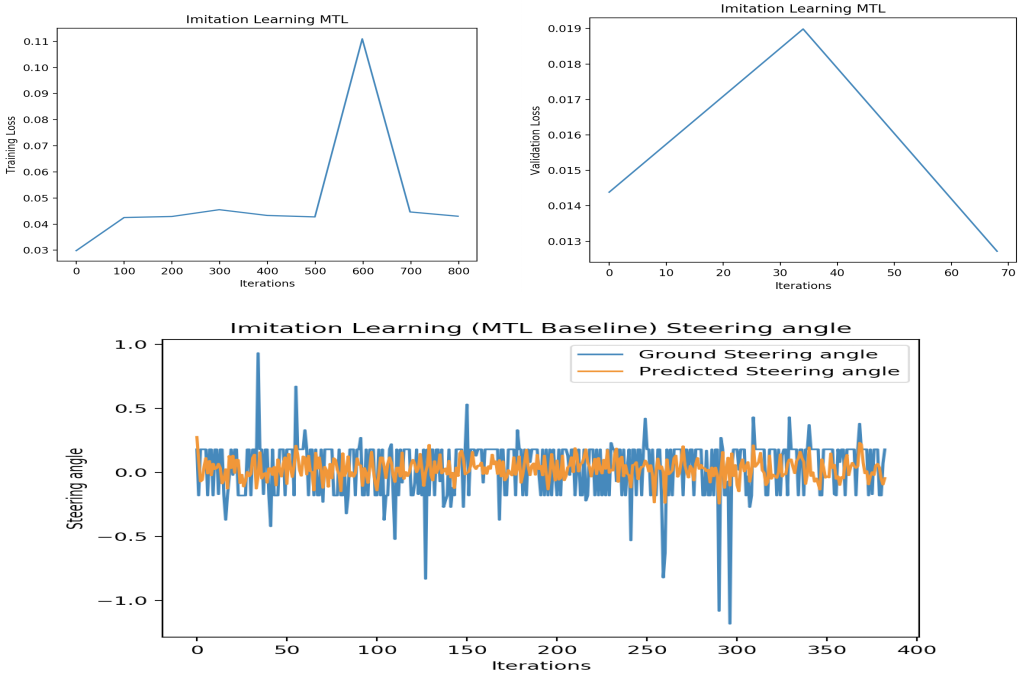



Figure 13: Training Loss, Validation Loss and Steering Angle for Multi Task Learning

### 5.4.2 Few-Shot Learning

The car manages to go till the bridge in the map, which is around 25% of the testing terrain. The reason for this is that it takes time for it to converge and might need a lot of training time. The current training time is not enough as there are a lot of images that are being passed and training takes place over the training set and the support set. Understandably, it might take time for the model to learn the features.

We included the training loss, validation loss and steering angle graphs. We also added the track representation of where the vehicles crash.

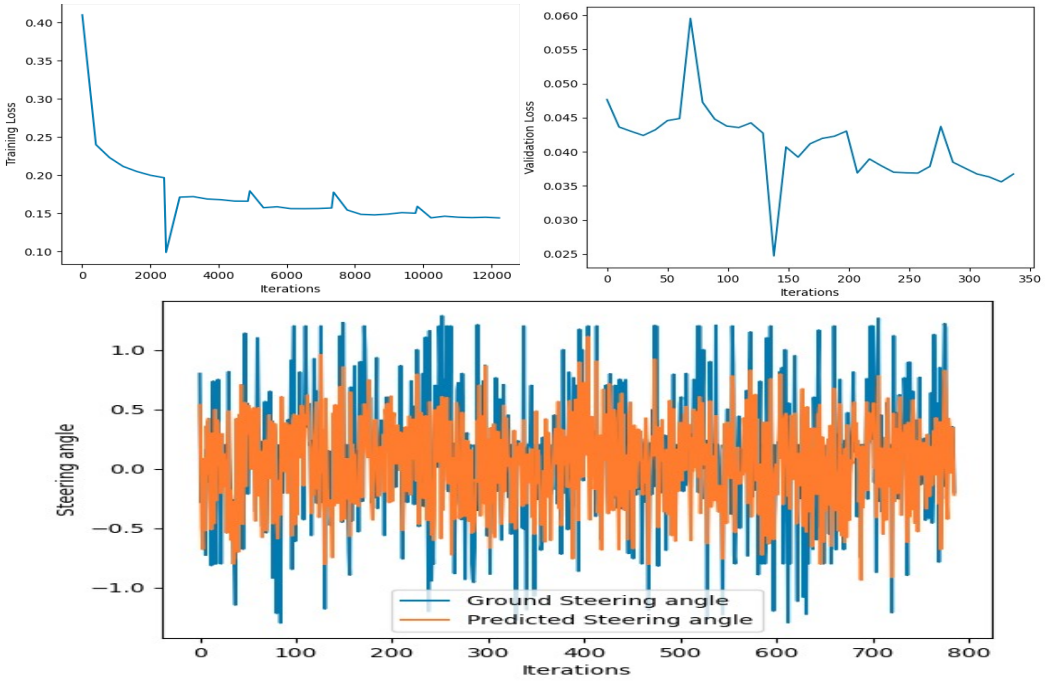



Figure 14: Training Loss, Validation Loss and Steering Angle for Few Shot Learning

From the graphs, we can observe that even though the training and validation loss decrease, the value of the loss is still high(0.15). Thus, the model has to train for a longer period.

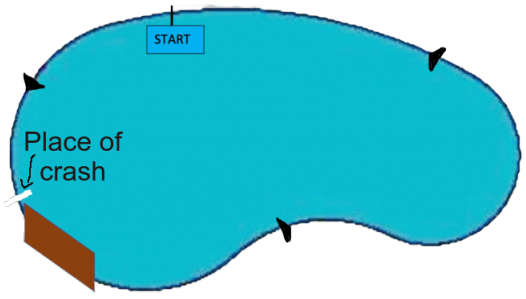



Figure 15: Place of Crash using Few-Shot Learning

## 5.5 Comparison

Both Deep Learning and Non-Deep Learning baselines perform well in case of steering control and complete the track. In the Deep Learning baseline, the vehicle moves without wobbling, but that

cannot be said of the Non-Deep Learning baseline. Thus, we can conclude that the deep learning baseline works the best. The multi-task model performs well with regard to speed control. It can maintain an average speed and slows down near turns. Our advanced deep learning baseline(few-shot learning) needs more computational resources for the training process to be able to perform well. The Track Completion Ratio is a metric defined to evaluate the models based on how much percentage of the testing track it covers.

| **Model** | **Completion Ratio (%)** |
|---|---|
| Non -Deep Learning Baseline | 100 |
| Deep Learning Baseline | 100 |
| Transfer Learning | 2 |
| Few-Shot Learning | 25 |
| Multi-Task Learning | 50 |
| Canny Edge Detection | 15 |
| Unsupervised Image Segmentation | NA |

The Validation loss is not a good metric for judging the performance of the car. Since this is a regression problem, we do not have accuracies to judge the performance of the model. Since this is also different from the traditional Reinforcement Learning techniques, there are no rewards given to the agent. Thus, we rely on the completion ratio as an effective metric to evaluate the performance of the models.

# 6 Discussion

One of the main limitations that we faced is the lack of proper compute power. Many of the simulators traditionally used for autonomous driving requires a lot of GPU computation. As a result, We had to choose a simulator with limited environments that required less GPU compute power. To make the model more robust, we need support data from numerous environments that are provided in other simulators that require higher GPU power. In the future, we are planning on extending this model with academic help to perfect the model. We also plan to add an Attention block to help with the weights. This would make it perform significantly better than the other models. So far, We have used the Multi-Task Learning model for speed control. We plan to extend this by integrating it with the Few-Shot learning model to get robust speed control and also adapt to different terrains without crashing anywhere. The final aim to make it run with obstacles on the road and train it to learn real-world scenarios.

# Acknowledgments

We would like to thank Prof.Konrad Kording for giving us a wonderful platform to showcase our Deep Learning knowledge. We would also like to show our gratitude to Jianqiao Wangni and Sadat Shaik for helping us over the various stages of the project. We are immensely grateful to the other Teaching Assistants for guiding us with the coursework throughout the semester.

# References

[1] Eric Zhan, Stephan Zheng, Yisong Yue, and Patrick Lucey. Generative multi-agent behavioral cloning. 03 2018.

[2] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning, 2017.

[3] Dean Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D.S. Touretzky, editor, *Proceedings of Advances in Neural Information Processing Systems 1*. Morgan Kaufmann, January 1989.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classificationwith deep convolutional neural networks. pages 1097–1105, 2012.

[6] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car, 2017.

[7] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving, 2015.

[8] Lu Chi and Yadong Mu. Deep steering: Learning end-to-end driving model from spatial and temporal visual cues, 2017.

[9] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception, 2018.

[10] Sauhaarda Chowdhuri, Tushar Pankaj, and Karl Zipser. Multinet: Multi-modal multi-task learning for autonomous driving, 2017.

[11] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[12] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation, 2017.

[13] Doyen Sahoo, Hung Le, Chenghao Liu, and Steven C. H. Hoi. Meta-learning with domain adaptation for few-shot learning under domain shift, 2019.

[14] A. Kanezaki. Unsupervised image segmentation by backpropagation. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1543–1547, 2018.

[15] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[16] Zhenye Na. Self-driving car simulator, 2019.

[17] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression, 1998.

[18] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[19] Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009.

[20] Asako Kanezaki. Unsupervised image segmentation by backpropagation. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 1543–1547. IEEE, 2018.

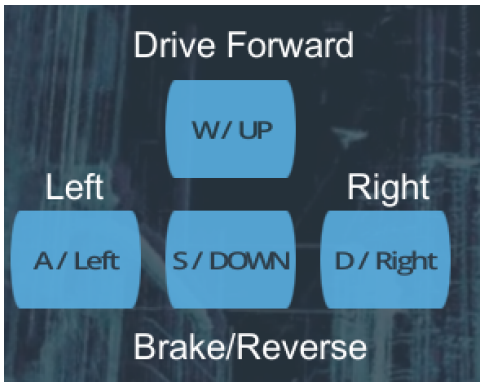
# 7 APPENDIX

## 7.1 Simulator & Dataset

We have used the Udacity Self-Driving Simulator to test our models. Udacity has extended to provide open access to this simulator as it was initially accessible only to people who register to the Udacity Nano-degree program. The graphics and tracks were simple and hence during training GPU consumption and computation would be less expensive. We thought it would be best to test our models related to behavioral cloning in this controlled environment.

Udacity Driving Simulator has two tracks- Desert and Jungle and two modes-Training and Autonomous mode. In the training mode, the user can drive the car manually to record the driving behavior. The generated images are saved locally and the paths of the images are mapped in a '.csv' file to steering angle,speed,brake and throttle values. The Autonomous Mode is used to test the model on one of the provided terrains. A good model is expected to perform well in the Autonomous Mode without any collisions.


Simulator




Controls

Simulator and its Controls

Desert Terrain

Jungle Terrain

Simulator Tracks

We decided to use an existing expert-driven dataset from Kaggle. The dataset was obtained from [16]. They provide free access to the dataset. It consists of images generated from the training mode. Each entry in the dataset consists of images from the left, right and center cameras placed on the bonnet of the moving vehicle. It also consists of continuous values- steering angle, speed, throttle and a discrete value brake. The steering angle and throttle are used as input to the simulator in the autonomous mode. Together they control the movement(direction and speed) of the vehicle in autonomous mode.

The Track1 (Desert) dataset contains 31,845 RGB labeled images consisting of 10615 images each for the center, left and right directions. The Track2 (Jungle) dataset contains 65,484 RGB labeled images with 21828 images each for the center, left and right directions. All images are resized to the resolution of $160 \times 320$ and are in JPG format. The three images (Center, Left, Right) at a particular instant are mapped to a particular steering angle, speed, brake force and throttle value.

| center | left | right | steering | throttle | brake | speed |
|---|---|---|---|---|---|---|
| IMG/center_2016_12_01_13_30_48_287.jpg | IMG/left_2016_12_01_13_30_48_287.jpg | IMG/right_2016_12_01_13_30_48_287.jpg | 0 | 0 | 0 | 22.14829 |
| IMG/center_2016_12_01_13_30_48_404.jpg | IMG/left_2016_12_01_13_30_48_404.jpg | IMG/right_2016_12_01_13_30_48_404.jpg | 0 | 0 | 0 | 21.87963 |
| IMG/center_2016_12_01_13_31_12_937.jpg | IMG/left_2016_12_01_13_31_12_937.jpg | IMG/right_2016_12_01_13_31_12_937.jpg | 0 | 0 | 0 | 1.453011 |
| IMG/center_2016_12_01_13_31_13_037.jpg | IMG/left_2016_12_01_13_31_13_037.jpg | IMG/right_2016_12_01_13_31_13_037.jpg | 0 | 0 | 0 | 1.438419 |
| IMG/center_2016_12_01_13_31_13_177.jpg | IMG/left_2016_12_01_13_31_13_177.jpg | IMG/right_2016_12_01_13_31_13_177.jpg | 0 | 0 | 0 | 1.418236 |
| IMG/center_2016_12_01_13_31_13_279.jpg | IMG/left_2016_12_01_13_31_13_279.jpg | IMG/right_2016_12_01_13_31_13_279.jpg | 0 | 0 | 0 | 1.403993 |
| IMG/center_2016_12_01_13_31_13_381.jpg | IMG/left_2016_12_01_13_31_13_381.jpg | IMG/right_2016_12_01_13_31_13_381.jpg | 0 | 0 | 0 | 1.389892 |
| IMG/center_2016_12_01_13_31_13_482.jpg | IMG/left_2016_12_01_13_31_13_482.jpg | IMG/right_2016_12_01_13_31_13_482.jpg | 0 | 0 | 0 | 1.375934 |
| IMG/center_2016_12_01_13_31_13_584.jpg | IMG/left_2016_12_01_13_31_13_584.jpg | IMG/right_2016_12_01_13_31_13_584.jpg | 0 | 0 | 0 | 1.362115 |

Dataset

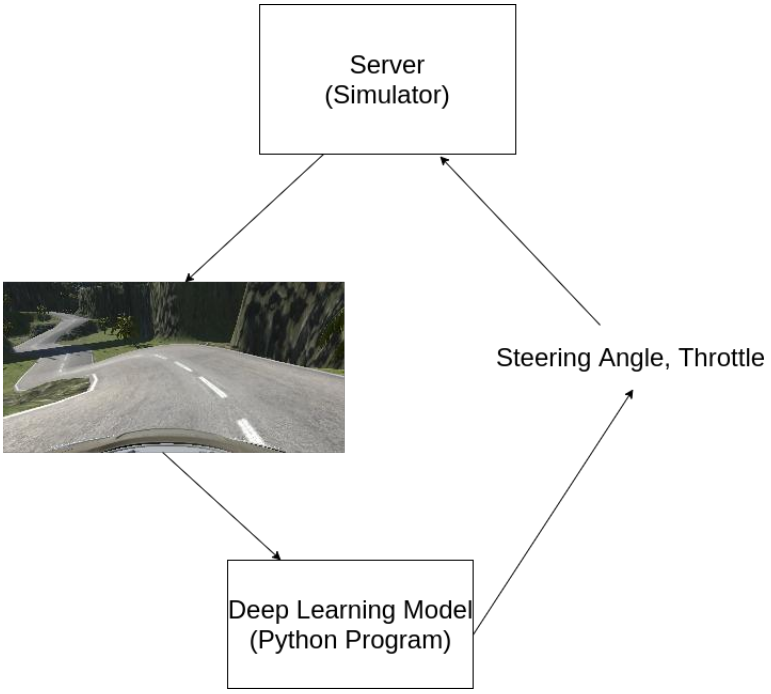



Figure 16: Telemetry

*Left Image *Center Image *Right Image

Left, Center and Right Images as in the Dataset

**TELEMETRY:**
We use Telemetry to communicate with the server. It gets the images along with the steering angle values from the simulator to the local python file. Once we get the predicted steering angle for the input frame, we return it back to the simulator along with the throttle. The skeleton code for telemetry(drive.py) came along with the simulator.

```
sio = socketio.Server()
def send_control(steering_angle, throttle):
    sio.emit(
        "steer",
        data={
            'steering_angle': steering_angle.__str__(),
            'throttle': throttle.__str__()
        },
        skip_sid=True)
```

## 7.2 Attempts taken to Improve Driving

**Canny Edge Detection:** Canny edge detection is a multi-step algorithm that can detect edges with noise suppressed at the same time. This was an attempt taken to perform domain adaptation across the two terrains. It uses Canny Edge Detection to only detect the lanes from the images. Therefore, the image would be converted into a black and white image with only lines representing the roads.This is

done to remove unnecessary features that the model can possibly learn. Initially, there is a conversion from RGB to Grayscale taking place. The Canny Edge Detection is then performed on the grayscaled image. This image is passed into the CNN for predicting the steering angle. This is done for training and testing images. The images coming from the simulator in the autonomous mode are also made to undergo the canny edge detection such that it resembles the training data.

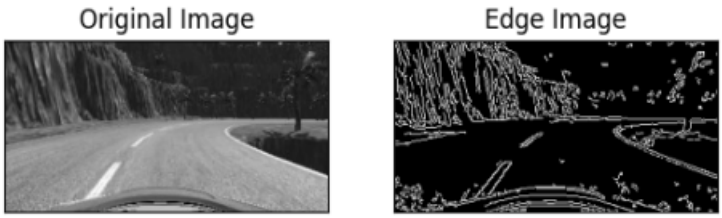



Figure 17: Canny Edge Detection

The edge image from the figure shows what is being passed into the CNN.

**Unsupervised Image Segmentation by Backpropagation:** The image segmentation process is unsupervised as there are no training images or ground truth labels of pixels that are given. The

Figure 18: Unsupervised Image Segmentation

reasoning behind the labelling is as follows:

- Pixels of similar features are assigned to be the same label
- Spatially continuous pixels are assigned as the the same label
- The number of unique labels is to be large.

This resulted in the segmented image that gave showed the road with a different colour. It is done to detect the desired feature from the image. It is expected to improve the performance while doing domain adaptation as it can select just the road from the image and remove all the unnecessary features from the image. This would help the model to learn the features better.

**Siamese Networks:** It is consists of two similar neural networks with each network taking one of the two input images. We use a contrastive loss function which calculates the similarity between the two images.
**Architecture:**

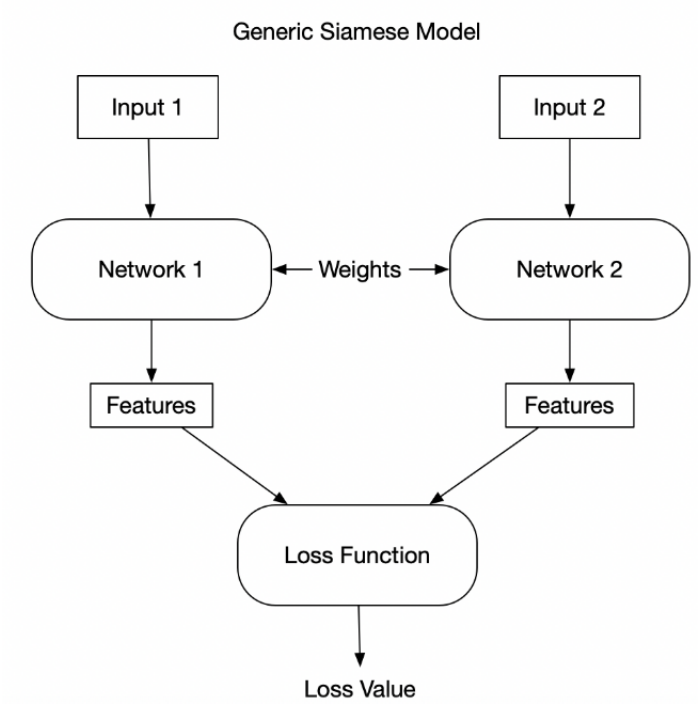



Figure 19: Architecture of Siamese Networks (taken from Medium)

From Figure 9, we see two images going into the respective networks and the loss being calculated

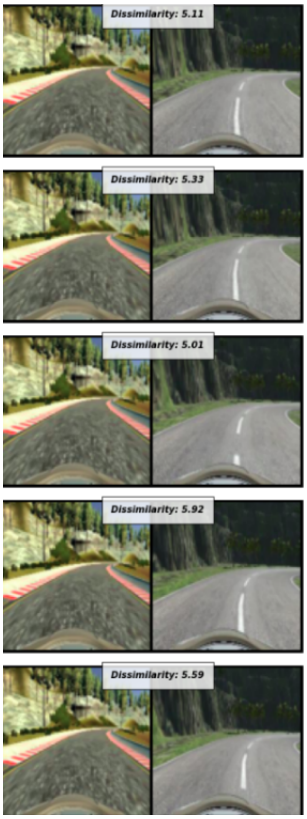



Figure 20: Siamese Networks

for both using the Constrastive loss. It is a Distance-based Loss function.
Let $x_1, x_2$ represent input 1 and input 2 respectively
The loss is calculated using the following formula:

$$L(W_1, W_2, x_1, x_2) = \frac{1}{2}(1-y)D^2 + \frac{1}{2}y\ max\{0, m-D\}^2$$

D = $|a-p|_2$ = $|f(W_1, x_1) - g(W_2, x_2)|_2$
y = 0 if $(x_1, x_2)$ if the pair is similar
y =1 if $(x_1, x_2)$ if the pair is dissimilar
m is the desirable distance for the dissimilar pair
This was an experiment done to find the similarity between the two terrains.

**HYPERPARAMETER TUNING:**
We used a learning rate of 0.0005 after trying out other learning rates in different ranges. The optimizer was chosen as Adam after it gave the best performance.

- Learning rate = 0.0005
- Batch Size = 8
- Loss = Contrastive Loss
- Optimizer = Adam

**Transfer Learning:** The Convolutional Neural Networks used for transfer learning tasks are networks that are pretrained using ImageNet[19] dataset that contains 1.2 Million images of belonging to approximately 1000 different classes. As a result, in these networks the initial layers are very efficient in recognizing generic features from the images and the layers at the end are more tuned to dataset specific tasks. Due to this property of these pretrained networks, they become ideal candidates

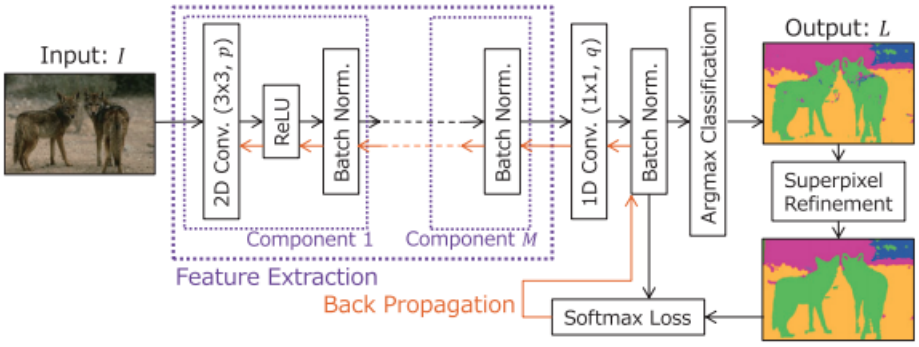



Figure 21: Unsupervised Image Segmentation from [20]

for transfer learning tasks where the learned parameters from the initial pretrained networks are transferred to a new model and the last layer of the model is finetuned with training to suit some other task. Thus, we experiment with these networks to see if they can generalize across domains without any help in a Zero-shot manner.

## 7.3 Analysis of the above Attempts

**Unsupervised Image Segmentation by Backpropogation** From the two images, we see that it eliminates the extra features and just gives the road in the new image. We make use of the Superpixel segmentation which divides an image into hundreds of non-overlapping superpixels. We calculate the Softmax loss between the responses from the network and also the cluster labels. From Figure 21, we see that the model learns by backpropagation. There is a forward and a backward pass. After the superpixel refinement, the softmax is calculated and is backpropagated through all the layers of the network back to the input image. We predict the cluster labels and train the network with the predicted cluster labels. Learning rate controls the balance between parameter updates and the clustering process. We used the model developed by Asako Kanezaki [20] in his work on unsupervised image segmentation.
However, we observed that it too around 2 minutes for each image to undergo the segmentation process. This is not feasible for the training process and especially the testing process as it could be time consuming.It would require very powerful computational resources for it to work. Therefore, we unanimously decided not to move ahead with this implementation.

**Canny Edge Detection:** Since there are sharp turns and some images where there are no lane markers, the Canny edge detection does not perform well. In fact, there are some areas where the lanes are not properly detected and gaps seen between the detected lines. This does not help in the performance as the car crashes at a sharp turn in the same terrain. This was trained and tested in the Desert terrain. This made us realize that this cannot be used for our implementation as it did not perform in the trained terrain itself.

**Siamese Network:** We have inferred that one-shot learning cannot be used as there is a difference between the left, right and the center frames of the two terrains. The model will not perform well with one image of the testing terrain. There arises a need to use more images of the testing terrain. Thus, we shift to Few-Shot Learning to make the model adapt well to the testing terrain.

**Transfer Learning:** ImageNet has a huge dataset of diverse images. Thus, two images of the road might appear similar to it. The minute differences in orientation of the car and the road might not be captured by the model. This could be the reason that the training and validation loss is high. We can also observe in the Steering angle graph that the predicted angle is between 0.5 and -0.5. This exDesert why the model crashes at the first steep turn.

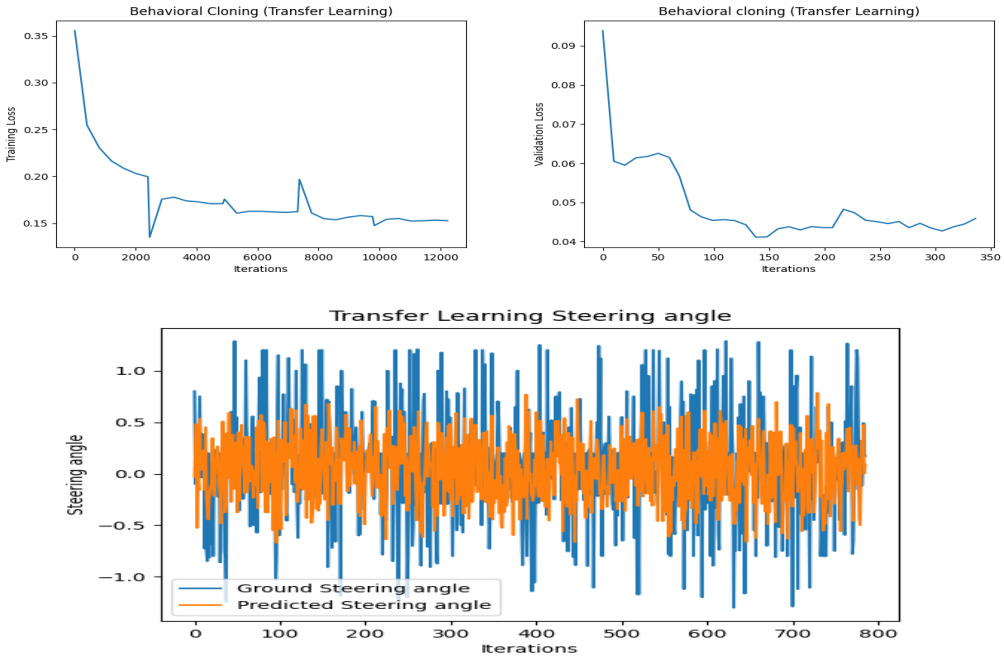



Figure 22: Training Loss, Validation Loss and Steering Angle for Transfer Learning