Skynet's Racetrack

by

Path to The Future

Ebrahim Rasromani

Zafir Momin

Eunjoo Ahn

Abstract

In this paper, we explore the application of imitation learning for autonomous driving. Our goal was to develop an autonomous driving agent in the Gym OpenAI "CarRacing-v0" simulated environment that can drive around the racing track close to human-level performance. We started with a state independent baseline model that simply drives straight. We then developed more sophisticated models with logistic regression, random forest, and convolutional neural networks. The average score of each model / agent across 15 episodes was used as the evaluation metric for model selection. The average score for the baseline was -16. A human expert yields an average score of 879. With convolutional neural networks, we were able to attain an agent that drives autonomously around the racing track at 98.6% human level performance.

Introduction

The OpenAI Gym is a Python based simulation environment that is used for imitation and reinforcement learning. Agents can learn from Gym's very diverse set of games such as ping pong, stacking blocks in three dimensional environments or classic Atari games. CarRacing-v0 is a birds-eye racing simulator of a race car on a track. Each 'state' or frame consists of 96x96 pixels by RGB image. On screen, there is a current reward score and four kinds of indicators. These indicators include the true speed of the car, four Anti-Lock-Braking System (or ABS) sensors, one for each wheel, steering wheel position, and a gyroscope sensor. This can be seen in Fig 1 below:

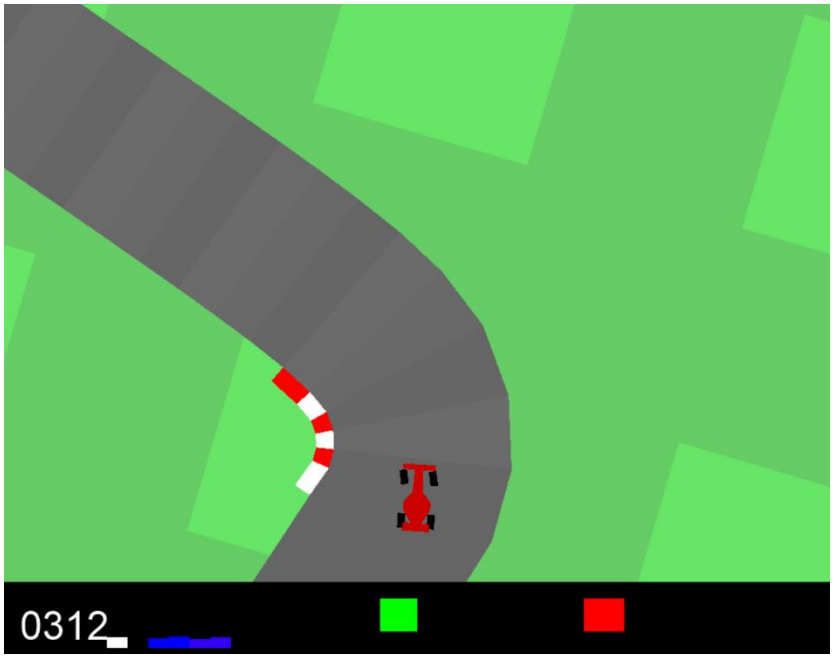


***Figure 1**: From left to right at the bottom of the screen: Reward score, True Speed (white), 4 ABS sensors (4 blue bars), Steering Wheel position (green) , and Gyroscope sensor (red).*

The reward system score is given by

$$\text{Reward} = 1000 - F/10$$

where 1000 is our maximum reward value for each track generated and F is the number of frames taken to complete one full lap. Once each tile of the track has been visited, which is equivalent to one lap, the game or "episode" ends, and the total reward is calculated. Note that a human expert yields a score of ~900 on average.

Business Understanding

Autonomous vehicles have been a central topic of discussion over the past few years. Their potential for reducing traffic deaths, emissions, and fuel consumption has propelled financial investments in that space. According to McKinsey's "The future of mobility is at our doorstep" 2019 article [1], autonomous vehicles are projected to generate a market revenue of $1.1 trillion from mobility services, such as ride-hailing, and $0.9 trillion from sales of autonomous vehicles by 2040. Currently, no fully autonomous vehicles exist but many companies such as Tesla and Waymo are working towards the development of such systems. Machine learning is an essential tool for the development of autonomous vehicles particularly for perception, planning, and control. Most current systems use imitation learning and neural networks for perception and control. Novel approaches using reinforcement learning are currently being researched to address edge cases with limited data.

In this paper, we will explore the autonomous vehicles space through an exercise of developing and deploying an autonomous driving agent in a 2D simulated environment. Our goal is to gain a basic understanding of imitation learning from which we can build upon in other subsequent projects to develop more advanced systems in 3D environments.

Data Understanding

Given that we are approaching the problem of developing an autonomous agent with imitation learning, we first needed to collect data from a human expert from which the agent can learn to imitate. This was done playing the OpenAI Gym “CarRacing-v0” game and collecting a screenshot of each frame and the actions taken at each step from ~16 episodes yielding a total of 20,000 steps. Each frame is a 96 by 96 pixel RGB image as shown in Fig 1, while each action is associated with a three element array as shown in Fig 3. The underlying data for the input to our models is the raw pixels from the frames and the output is the associated action. Given that the action space is discrete, this is a classification problem. Note that our dataset of 20,000 state-action pairs was split into a training and testing set such that the first 90% of the data is associated with the training set, while the last 10% is associated with the testing set.

Data Preparation

The dataframe used in processing comprises the image data as the features and the action taken in that frame as the target or label for that frame. As such, the actions and the individual frames must be extracted from the recorded data and formatted to create our dataset. Since the data was recorded by our team, the extent of preparation was rather minimal as compared to traditional data cleaning and processing methods. First and foremost, the screen capture frames had to be downsized with regards to dimension so that they could be processed. The RGB images collected were reduced from a dimension of 96x96x3 to 96x96 when they were converted to grayscale. For the non-deep learning models, each frame matrix was flattened to a single row vector of length 9216 (96 times 96). The process is shown in a simple diagram in Fig 2. Given that the deep learning models are associated with convolutional neural networks, the input data was left as a matrix to preserve the spatial relationship between pixels.

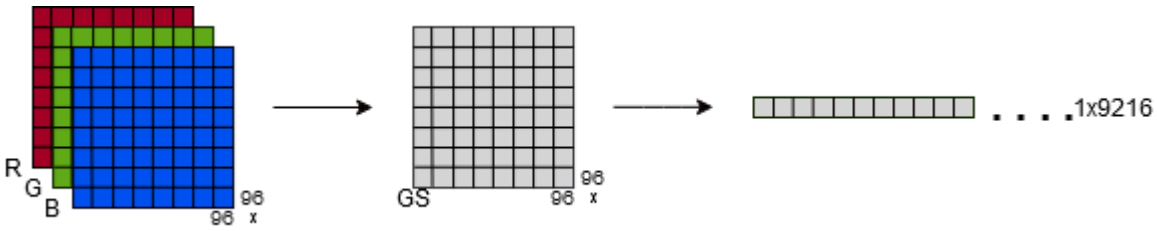


***Figure 2**: A single screenshot in RGB is turned into grayscale (denoted GS) to reduce dimensionality. The frame is then stored as a row vector. This process is repeated for each of the stored frames.*

Next, we work on labeling each of these rows/frames with their respective action. The 'action' section of our collected data is given in the form of [turn, accelerate, brake]. Using our settings for each action we can provide a label to each action. These labels range from 0 to 8 for a total of 9 classes. These are shown below in Table 1.

Action	Name	Label
[0 , 0 , 0]	Rest	0
[0 , 1 , 0]	Accelerate	1
[0 , 0 , 0.2]	Brake	2
[1 , 0 , 0]	Right	3
[-1 , 0 , 0]	Left	4
[1 , 1 , 0]	Right Accelerate	5
[1 , 0 , 0.2]	Right Brake	6
[-1 , 1 , 0]	Left Accelerate	7
[-1 , 0 , 0.2]	Left Brake	8

***Table 1**: Each setting on the car for [turn, accelerate, brake] corresponds to action the car takes. These are then given a class label which will serve as our target variable.*

Once we have a label for each frame the labels are then appended to the first column of each row thus completing our dataframe. Using Sklearn the data was then split into train and test sets on which the models were fitted and then evaluated respectively. The dataframe consisted of a little over 9200 columns, or features, and 20,000 rows with each row given a label from 0 through 8

for the 9 distinct classes. Each ‘datapoint’ or value in each cell, contained a value from 0 to 255. All input data was normalized to zero mean and unit variance. Below is data distribution for the labels / target variable.

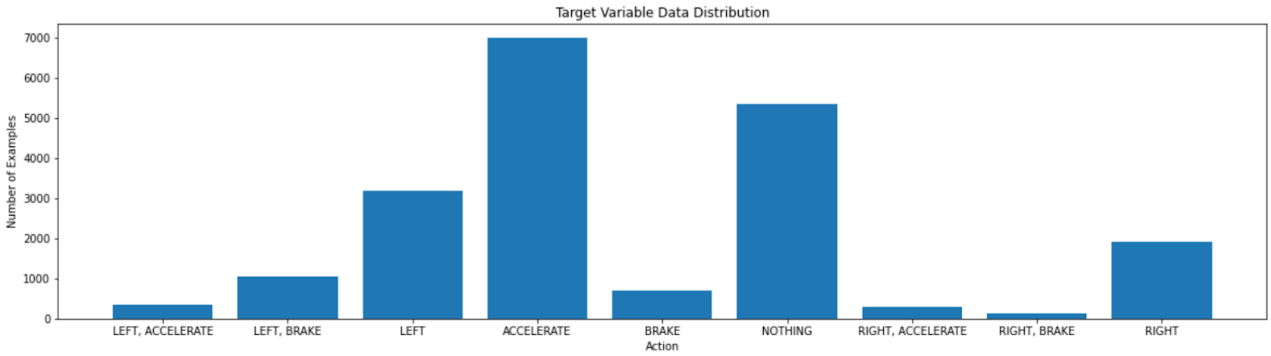


***Figure 3**: Target variable data distribution*

Modeling

Logistic Regression and RF:

We decided to use a logistic regression, hereby denoted as LR, as our first model as they are simple and easy to understand. Seeing that we are dealing with 9 classes, we employed Sklearns OneVsRest Classifier, on top of our LR model, which is a method of using binary classification for multi-class classification. The figure below shows the confusion matrix for our LR model.

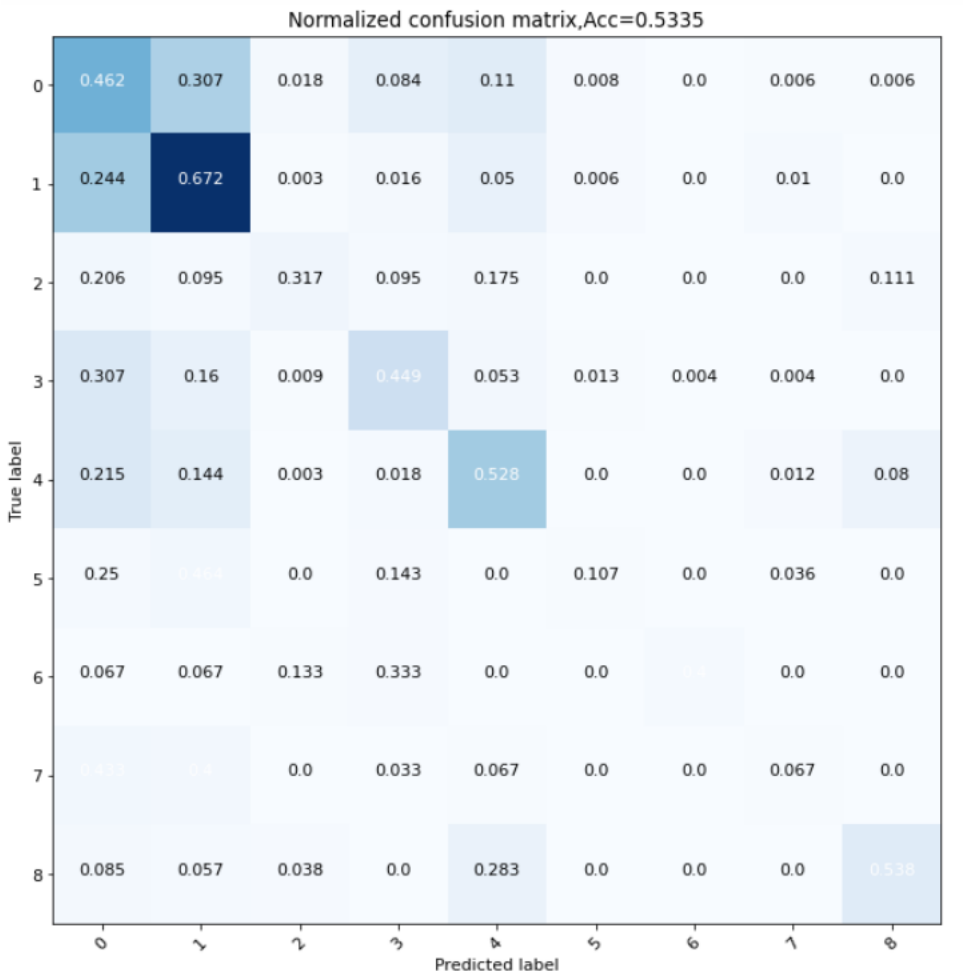


***Figure 4**: The confusion matrix for the Logistic Regression model. It has an accuracy of 53.35%*

The LR model returned an accuracy of approximately 53% and we expected this model to perform quite poorly. Driving a car with a 53% accuracy in your actions would be catastrophic and we quickly realized we need a different, more accurate model. For our next model we chose a random forest, hereby denoted RF, model. RF models are known to be excellent classifiers and perform well under large datasets. We employed a RF model (n=200) and the confusion matrix is shown below.

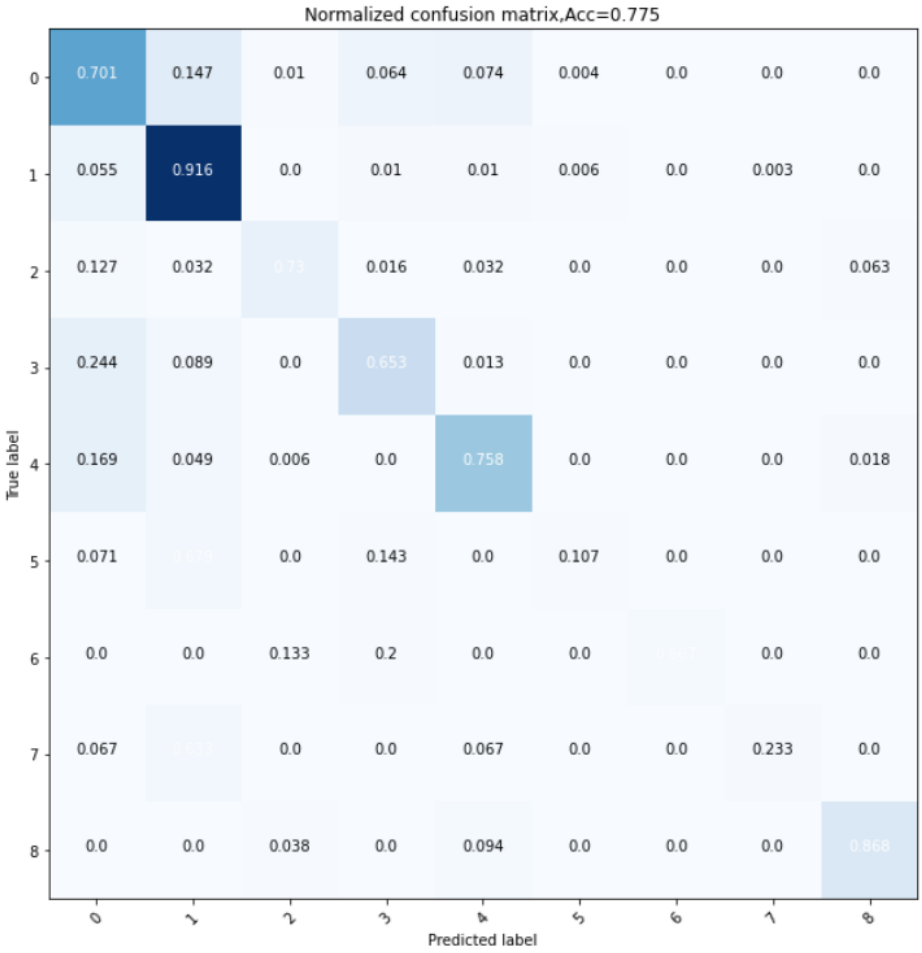


Figure 5*: The confusion matrix for the Random Forest model. It has an accuracy of 77.5%*

The RF model boasts an accuracy of 78.5% and performed significantly better during evaluations as detailed in the next section.

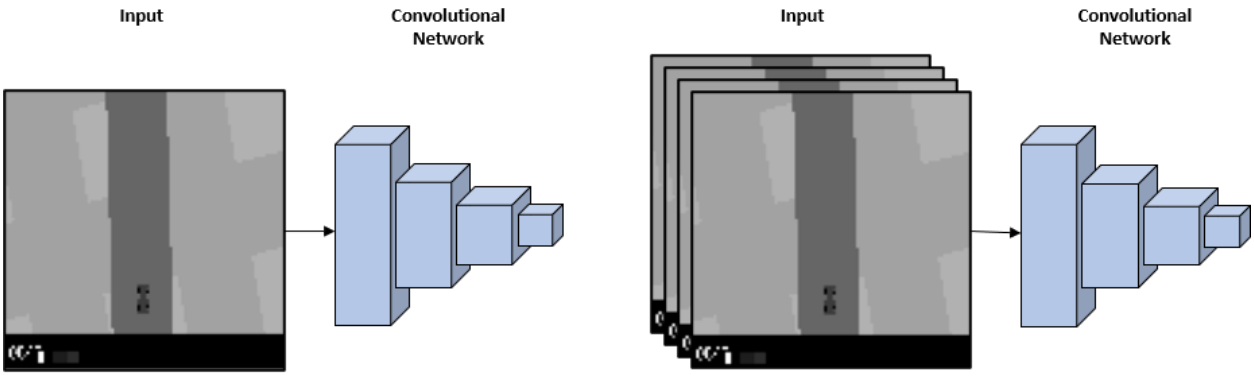
To provide insight into a possible reason that the models do not have a higher accuracy we can study the confusion matrices. A key point to note in the confusion matrices is that the labels that the model predicts incorrectly have sizable errors where there is a possibility that the labels overlap. For example, the true label of 5 which corresponds to the action of right and accelerate has a very high prediction rate for the label of 1 which corresponds to accelerate. Theoretically speaking, there are an infinite number of ways to make the turn around the same corner. As such the racecar/model is predicting it should accelerate where it should accelerate

and but also turn right. This is a result of having very labels that are very closely related. Possible next steps could entail exploring a clearer separation of labels so that they are independent to prevent this kind of error. Though we ran short on time, we would also like to further refine our LR and especially our RF models by performing grid search to optimize our hyperparameters and thereby maximizing model performance.

Neural Net:

Given that we are working with image data, we have decided to use convolutional neural networks for our model. Convolutional neural networks have a strong prior of what is known as locality and *stationarity* that make them a good feature extractor for images. Locality assumes that input signals are local such that neighboring points are more correlated than points far away from each other. Stationarity assumes that the input signals are stationary such that features detected by a filter can occur in many parts of the input domain. These two assumptions are applicable to the type of data we are using for our problem.

There are many possible choices for the structure of our neural network architecture. We have narrowed it down to 4 options that ultimately dictate the form of the input and the output for our model as shown in the figures below.



(a) (b)

***Figure 6**: Options for neural network architecture that dictate the form of our model input. (a) single channel input of one grayscale image. (b) multi-channel input of a stack of grayscale images.*

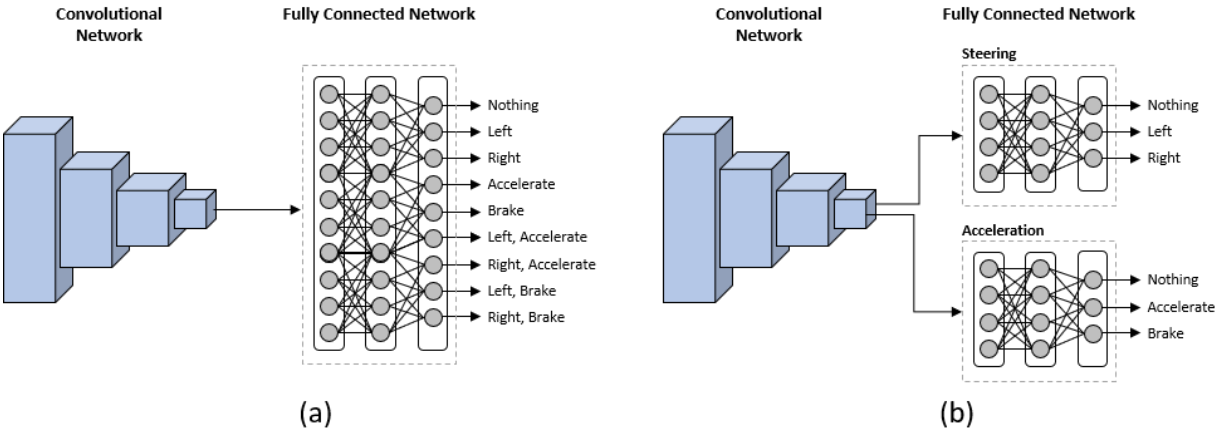


***Figure 7**: Options for neural network architectures that dictate the form of our model output. (a) convolutional neural network feature extractor followed by a fully connected network classifier that outputs 9 classes. (b) convolutional neural network feature extractor followed by two fully connected network classifiers, one for steering and one for acceleration, that output 3 classes each*

For our input, we can either have a single grayscale image or a stack of grayscale images. Ultimately, the choice of input will dictate the number of input channels into our convolutional neural network and will therefore dictate the kernel width of the first layer of our network. A single grayscale image is easier to process, as the OpenAI Gym "CarRacing-v0" environment feeds one image at a time and would yield a slightly less computationally intensive forward and backward pass. With a single image, however, the network will not be able to extract features associated with velocity, which we suspect is an important feature for selecting an action. A stack of subsequent frames will however require an additional step to process the frames and concatenate them during model deployment and may yield a slightly more computationally intensive forward and backward pass. However, the additional amount of data provided to the

network may yield significant performance improvements to justify the added complexity of having multiple frames as an input.

Note that our action space ultimately has 9 classes as shown in Fig 7a. However, the action space was constructed from a combination of two separate actions, one for steering and one for acceleration, as shown in Fig 7b and yields 6 classes in total. We therefore have two options for representing our output which will dictate the form of our network architecture. With 9 classes, only one fully connected network is needed following the convolutional feature extractor which yields a smaller, less computationally expensive network. The disadvantage is that the network now has a much harder task of correctly selecting one of nine possible classes. With 6 classes, two fully connected networks are needed following the convolutional feature extractor, one network to classify an action for steering and another to classify an action for acceleration. This approach yields a larger, more computational expensive network. Another disadvantage of such an approach is that two separate loss functions are needed for the output of each network and the way of combining the two loss functions into one is non-trivial. The advantage of this approach is that each fully connected neural network can specialize, one network specializes in classifying steering actions while another network specializes in classifying acceleration actions. The other advantage is that now each network has a much easier task of correctly selecting one of three possible classes.

	Output of 9 classes	**2 outputs of 3 classes each**
Input of one image	*Model A*	*Model B*
Input of stack of five images	*Model C*	*Model D*

Table 2: *Models A-D with their respective settings*

Throughout the rest of this document, we will refer to each of the four options for our models as *Model A-D* as shown above. The architecture of each of our models boils down to a convolutional neural network feature extractor followed by a classifier. The feature extractor consists of 4 convolutional layers, each made up of 2D convolution followed by a relu non-linearity and batch-normalization. Each layer consists of 16, 32, 64, and 64 channels respectively. The kernel size for all convolutions is 3 by 3 except for the first layers which has a size of 5 by 5. Similarly, the stride of all convolutions is 2 except for the first layer which has a stride of 1. All features outputted by the feature extractor are flattened into a vector before being fed into the classifier. *Model A* and *Model B* have the same convolutional feature extractor component with a 1 channel input. *Model C* and *Model D* also have the same convolutional feature extractor component but with a 5-channel input. *Model A* and *Model C* have the same classifier component. The classifier consists of a two-layer multi-layer perceptron with a relu non-linearity, 50 hidden activations, and 9 output activations as shown in Fig 7a. *Model B* and *Model D* also have the same classifier components. The classifier consists of two subnetworks, one for steering and one for acceleration. Both subnetworks are fed the same input from the feature extractor and each have the same architecture. The architecture consists of two layer

multi-layer perceptron with a relu non-linearity, 50 hidden activations, and 3 output activations as shown in Fig 7b.

The loss functions used for *Model A* and *Model C* is the cross-entropy loss function commonly used for classification problems. For *Model B* and *Model D* we use a loss function given by the weighted addition of two cross-entropy loss evaluated following each subnetwork and is expressed as follows

$$L = \lambda L_{steer} + L_{accelerate}$$

where L, L_{steer}, and $L_{accelerate}$ is the total loss, cross-entropy loss for the steer subnetwork, and the cross-entropy loss for the accelerate subnetwork. λ is a scaling factor which is set empirically to ensure each component of the loss are of similar scale. Based on our observations, yields comparable scales between the steering and acceleration loss components.

To train our models, we used Adam optimizer with learning rate selected via the learning rate finder method as described by Smith[3]. Below is an example figure of the learning rate finder process. The point at which the training loss decreases at the highest rate yields an appropriate learning rate.

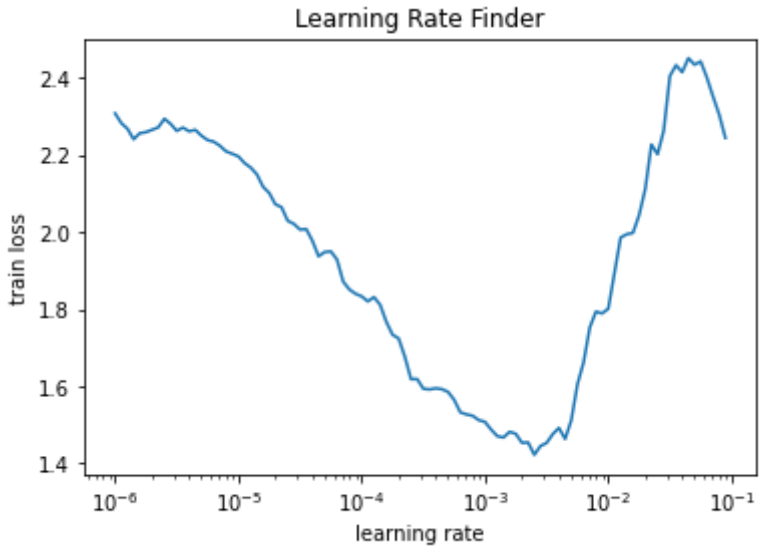
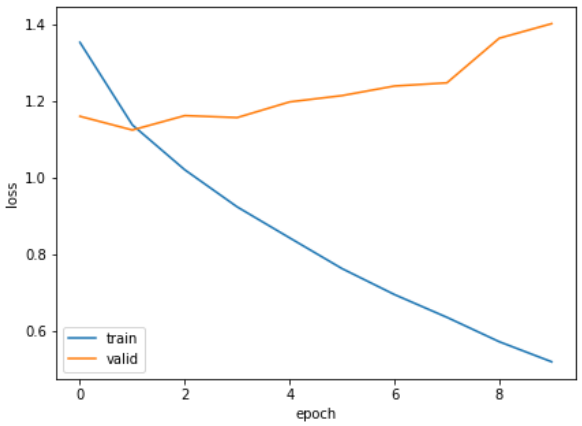
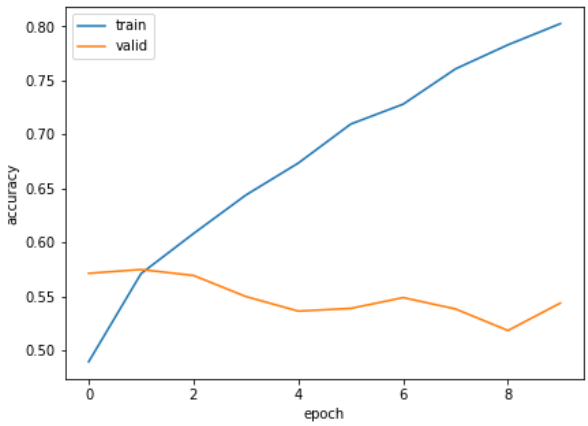


***Figure 8**: Learning rate finder used to find an appropriate learning rate for training*

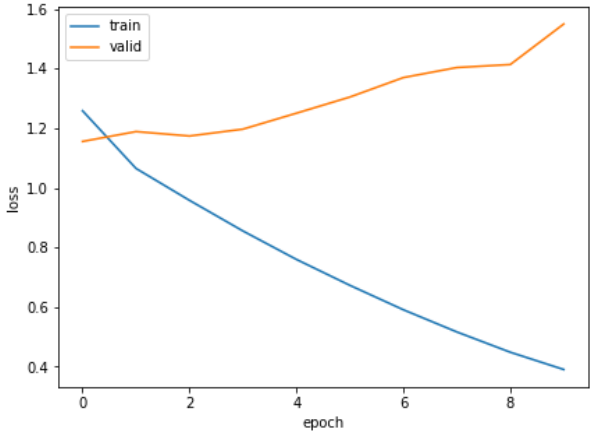
Note that throughout model training, we monitored that model performance by measuring the loss and the accuracy for the training set and the validation set. Accuracy is defined as the percentage of instances that the model correctly classified. Each model was trained over 10 epochs and the performance across epochs is shown below. Note that the performance curves are indicative of overfitting and that more data is needed to help our model generalize better. Unfortunately, our limited RAM capacity did not allow for a larger dataset. If we had more time, we would have saved the data in memory via images and loaded the images into RAM one at a time throughout training.

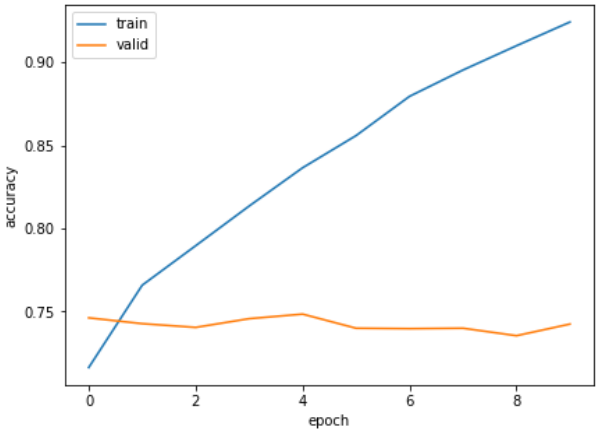
Model A Performance



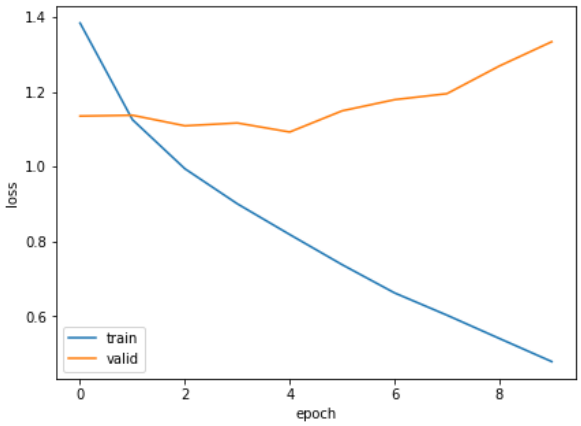


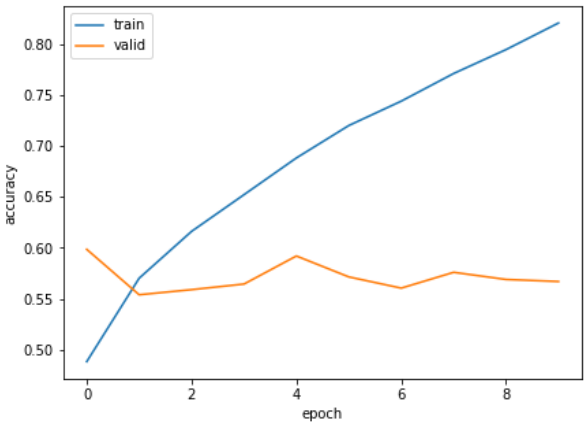
Model B Performance





Model C Performance





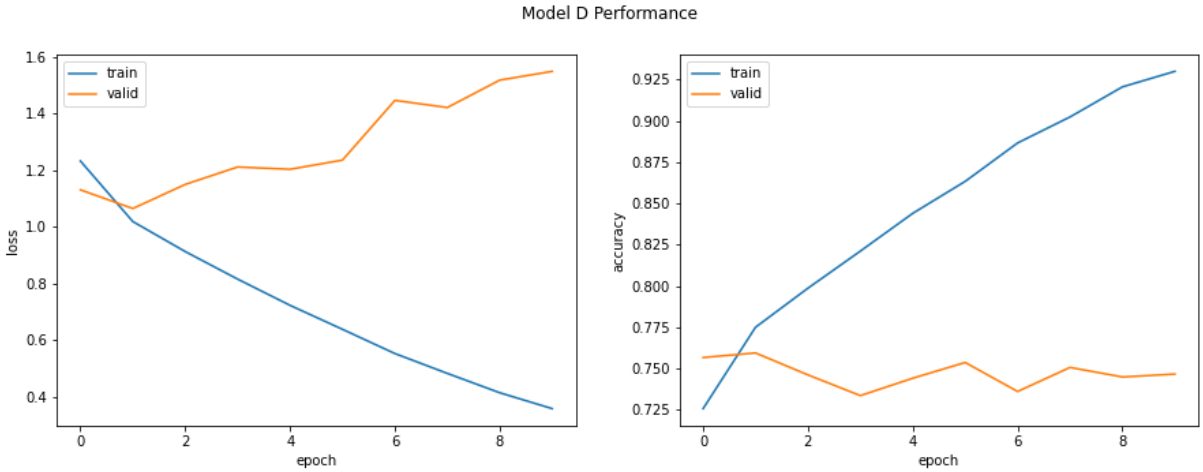


***Figure 9**: Model performance across epochs for the four models we selected*

At first glance, *Model B* and *Model D* seem to perform best in terms of accuracy. However, we need to fully evaluate the model using our evaluation metric for model selection of average game score over 15 episodes.

Evaluation

To evaluate our models, we first developed a baseline model. Our baseline model outputs the "accelerate" action regardless what state the agent is in. Our evaluation metric for model selection is the average score of the agent in the OpenAI Gym "CarRacing-v0" environment after 15 episodes. Ultimately, the evaluation metric was selected as it is closely aligned with the objective of our problem which is to deploy an autonomous agent that yields a high score in the "CarRacing-V0" environment. The average score of our baseline model is -16 with a standard deviation of 25. Note that human level performance yields an average score of 879 with a standard deviation of 58. Therefore, our goal was to find a model that performs somewhere between the baseline model performance (-16 average score) and the human level performance (879 average score). Below is the average game score over 15 episodes for each model developed.

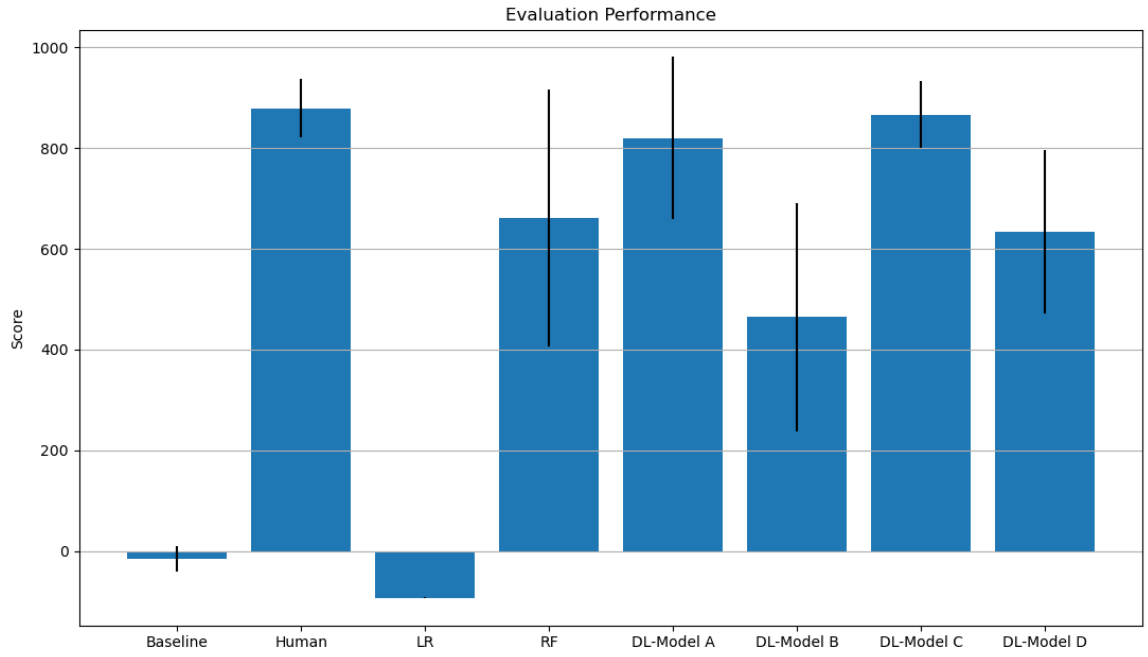


***Figure 10**: Comparison of model final performance measured in terms of average score over 15 episodes*

Model	Avg. Score (rounded)	Std. Dev
Baseline	-16	25.08
Human	879	58.31
LR	-93	0.58
RF	660	255.41
DL-Model A	820	161.35
DL-Model B	464	225.93
DL-Model C	867	66.85
DL-Model D	633	162.18

***Table 3**: The evaluated models are listed with their average scores and their standard deviations*

As expected from the low accuracy, the LR model performs quite poorly with an average score of -93, thereby underscoring the baseline and showing that the model is not viable. The RF model seems to exceed expectations with an average score of 660. However, it is to be noted that it also has the highest standard deviation of all models. It is clear that *Model C* yields the best

performance in terms of both mean and standard deviation of the score over 15 episodes. The relatively low standard deviation indicates that the model can consistently perform well. In fact, *Model C* performs at 98.6% human level performance and yields a similar variance as a human player. Recall that our objective is to develop an agent that can autonomously drive the car around the track in the Open AI Gym "CarRacing-v0" environment. *Model C* is able to achieve this objective by performing very close to human level performance consistently.

Deployment

We finally have a model that can autonomously drive the car around the track in the OpenAI Gym "CarRacing-v0" environment. During model evaluation, we effectively deployed the model in the racing environment to access how it would perform and the average score of 15 episodes was used as the criteria for model selection. In an actual production system, we would need to evaluate the performance of the model over-time and would ideally implement a DAGGER like system described by Ross[2]. The biggest issue with the imitation learning approach we have implemented is that if the model performs actions that lead it to be in a state that it has not seen before, then it will be difficult for the model to know how to behave in that scenario. E.g., race car goes far off track and drives in circles. Given that our input data was collected by a human, the training data distribution mostly consists of data in states in which the agent is on the track or sometimes slightly off the track. Once the agent enters a state far from the training data distribution, such as completely off the track, the agent will not be able make the appropriate decision to get back on the track. One way to address this issue is to monitor the incoming data distribution of the frames and estimate the likelihood that the incoming data comes from the training data distribution. If the likelihood is low, then our system should get input from a human expert how to behave in that situation. The human input will be collected

and integrated into the training data. Upon retraining, the system will be able to learn how to behave in a more diverse set of situations. Once such a system is deployed in real autonomous vehicles, one must consider many ethical concerns. Such concerns include whether the car should prioritize the safety of its passenger over pedestrians and whether customer driving data can be used by autonomous vehicle companies to dictate the insurance premiums.

Works Cited

1. Moller, Timo, et al. “The Future of Mobility Is at Our Doorstep.” *McKinsey & Company*, McKinsey & Company,19 Dec. 2019, www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-future-of-mobility-is-at-our-doorstep.

2. Ross, Stéphane, et al. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.” *Arxiv.org*, Carnegie Mellon University, 16 Mar. 2011, arxiv.org/pdf/1011.0686.pdf.

3. Smith, Leslie N. “Cyclical Learning Rates for Training Neural Networks.” *Arxiv.org*, U.S. Naval Research Laboratory, Code 5514, 4 Apr. 2017, arxiv.org/pdf/1506.01186.pdf.

Appendix:

Ebrahim Rasromani - Data collection, Data Exploration, Meeting Supervisor, Deep Learning Modeling, Figures and images, Github version control, Report writing, Code Debugging

Zafir Momin - Data collection, Data Exploration, Supervised Modeling, Figures and images, Github setup, Report writing, Code Debugging

Eunjoo Ahn - Meeting minutes, Data Visualization, Modeling. Code Debugging, Report Writing