F1 RACE WINNER PREDICTOR

MSAI 349

Fall 2023

Motivation

F1 is an exciting sport that millions watch. There is also a betting market. Our models should predict the winner of a race in a manner in which a gambler could place a bet before the winner finishes the race.

Objective

Build models that based off information about the first set of laps predict the winner of the F1 races.

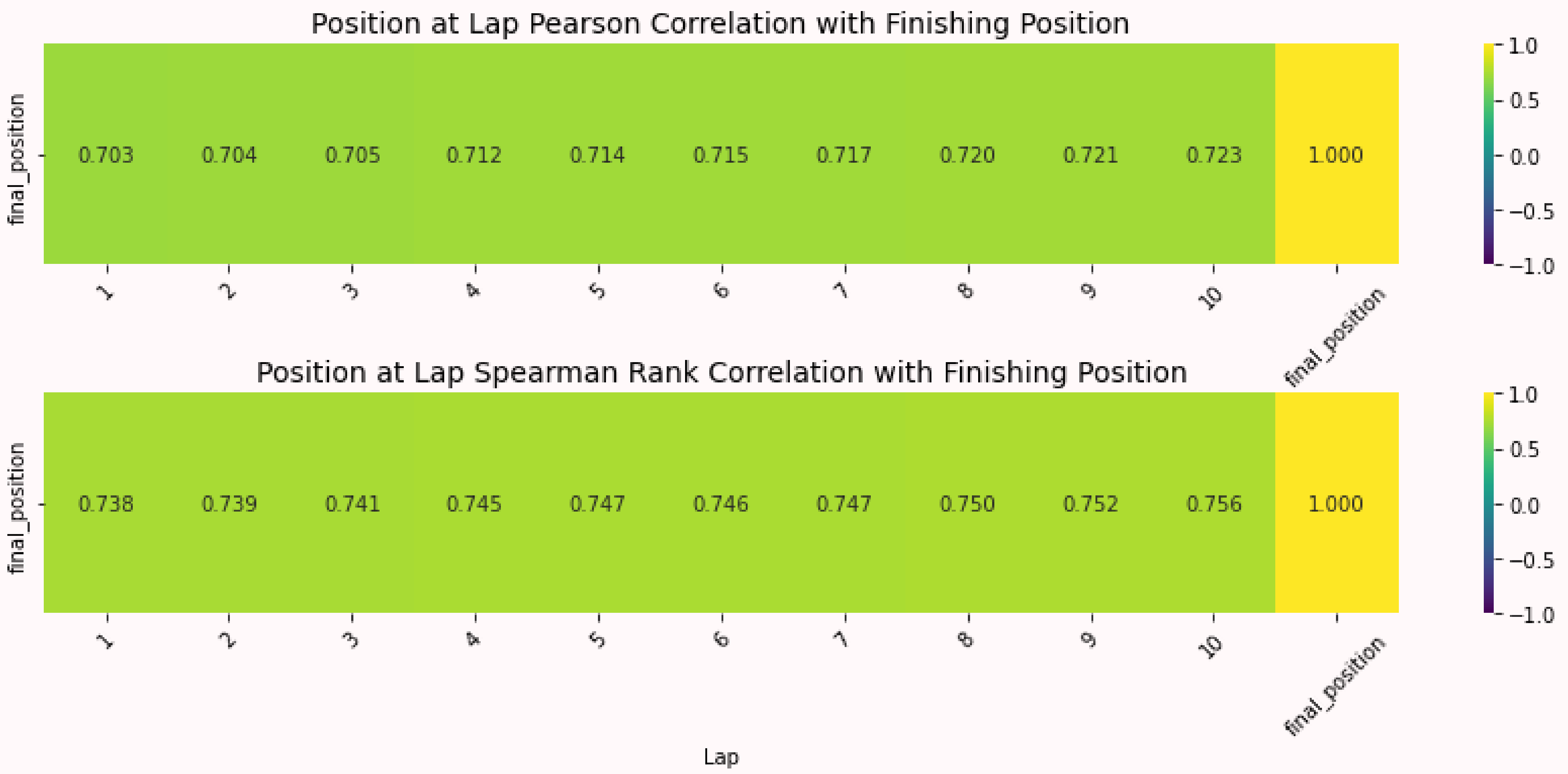
Data Source

kaggle

A collection of 13 datasets built from the Ergast API on F1 races from the beginning of F1 world championships (1950) to 2023

- Race results for 1091 races
- Valid race results for races since 1964 (960 races)
- Lap times for 510 (valid) races
- Data spans information about the race (circuit, country, etc.), drivers, constructors (cars), race results, lap times and pitstops

Background



Baseline Model

Predict finishing position based off positions at lap *n*

First Place Accuracy Table

	lap	train	valid	test
0	Lap1	0.563725	0.470588	0.509804
1	Lap2	0.552826	0.549020	0.549020
2	Lap3	0.555283	0.568627	0.568627
3	Lap4	0.557740	0.568627	0.588235
4	Lap5	0.577396	0.568627	0.607843
5	Lap6	0.577396	0.588235	0.607843
6	Lap7	0.570025	0.588235	0.607843
7	Lap8	0.567568	0.588235	0.607843
8	Lap9	0.574939	0.568627	0.607843
9	Lap10	0.582310	0.588235	0.588235

Model Diagram

X

race 1 randomized copy 1	**driver 1 pos after lap 1**	**driver 1 pos after lap 2**	**...**	**driver 24 pos after lap 10**
race 1 randomized copy 2			...	
race 2 randomized copy 1			...	
...	...	...	...	

Regression

y

race 1 randomized copy 1	**driver 1 finishin g position**	**driver 2 finishign position**	**...**	**driver 24 finishing position**
race 1 randomized copy 2			...	
race 2 randomized copy 1			...	
...	...	...	...	

Classification

argmin(y)

race 1 randomized copy 1	**index of winning driver**
race 1 randomized copy 2	index of winning driver
race 2 randomized copy 1	index of winning driver
...	...

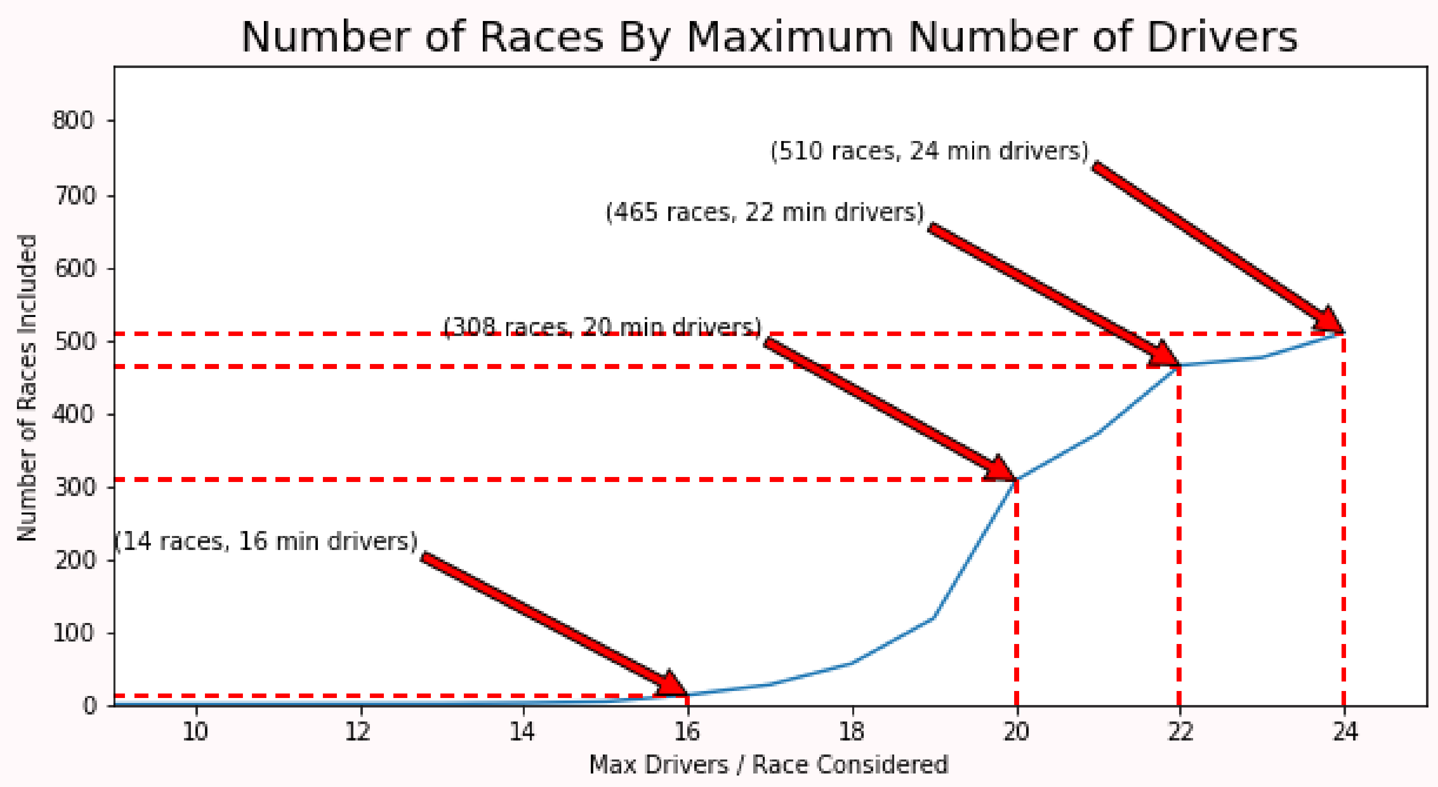
Example FFNN Architectures

```
class TwoHiddenLayerReg(nn.Module):
    def __init__(self, input_size=240):
        super(TwoHiddenLayerReg, self).__init__()
        self.fc1 = nn.Linear(input_size, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 24)
```

```
class OneHiddenLayerClassification(nn.Module):
    def __init__(self):
        super(OneHiddenLayerClassification, self).__init__()
        self.fc1 = nn.Linear(240, 128) # Input layer
        self.fc2 = nn.Linear(128, 64)  # Hidden layer 1
        self.fc3 = nn.Linear(64, 24)   # Output layer (24 positions)
```

Data Padding, Imputation, Randomization

Padding vs. Data Size Tradeoff



- Padding
  - Used 24 drivers in model (datapoint length = 24 * n laps)
  - Filled made-up drivers with position=25 for all laps
- Imputation
  - For drivers who started but didn't finish, their invalid laps were also filled with 25
  - Target for driver's that didn't finish = 25

Randomization

- The order of the drivers as they appear in a single data point (feature set row) were randomized
- 3 randomized copies of each race were made to generate more data

Data Sizes

- Train: 408 -> 1224
- Valid: 51 -> 153
- Test: 51 -> 153

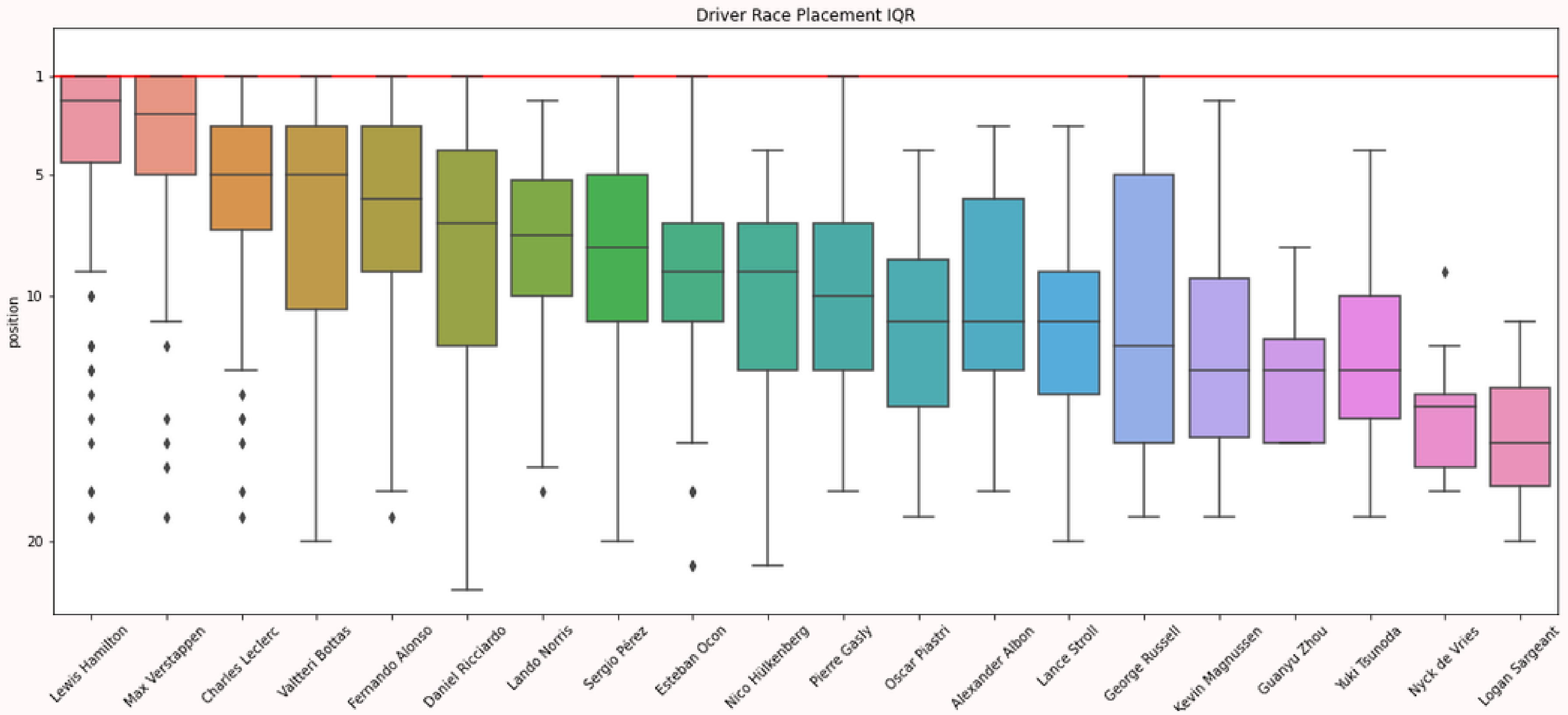
Add Feature: Associate Laps with Driver Information

Create a Performance Metric for Each Driver

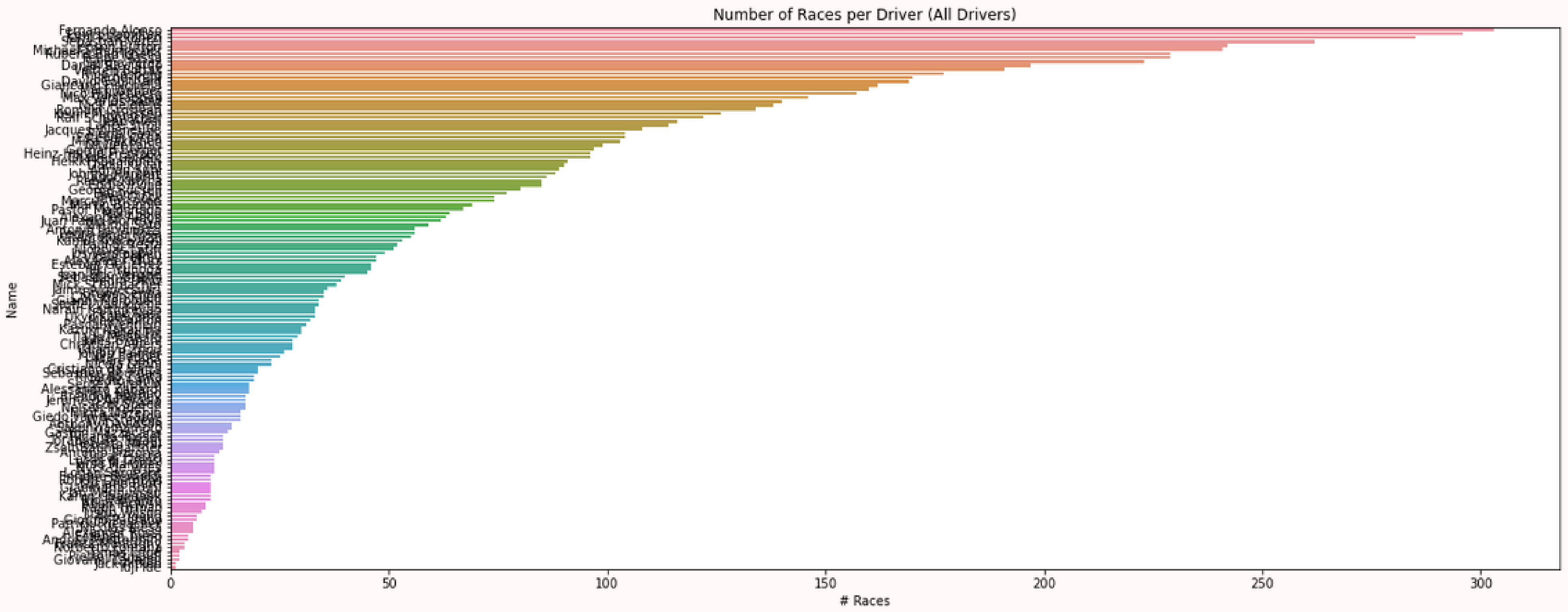
- Use weights from a linear regression
  - X = OHE drivers, y = finishing place
- Higher coef means the driver is associated with higher finishing position
- Add more variables to measure driver performance, holding other factors constant

Considerations with the Driver Feature

Variability in driver finishing position

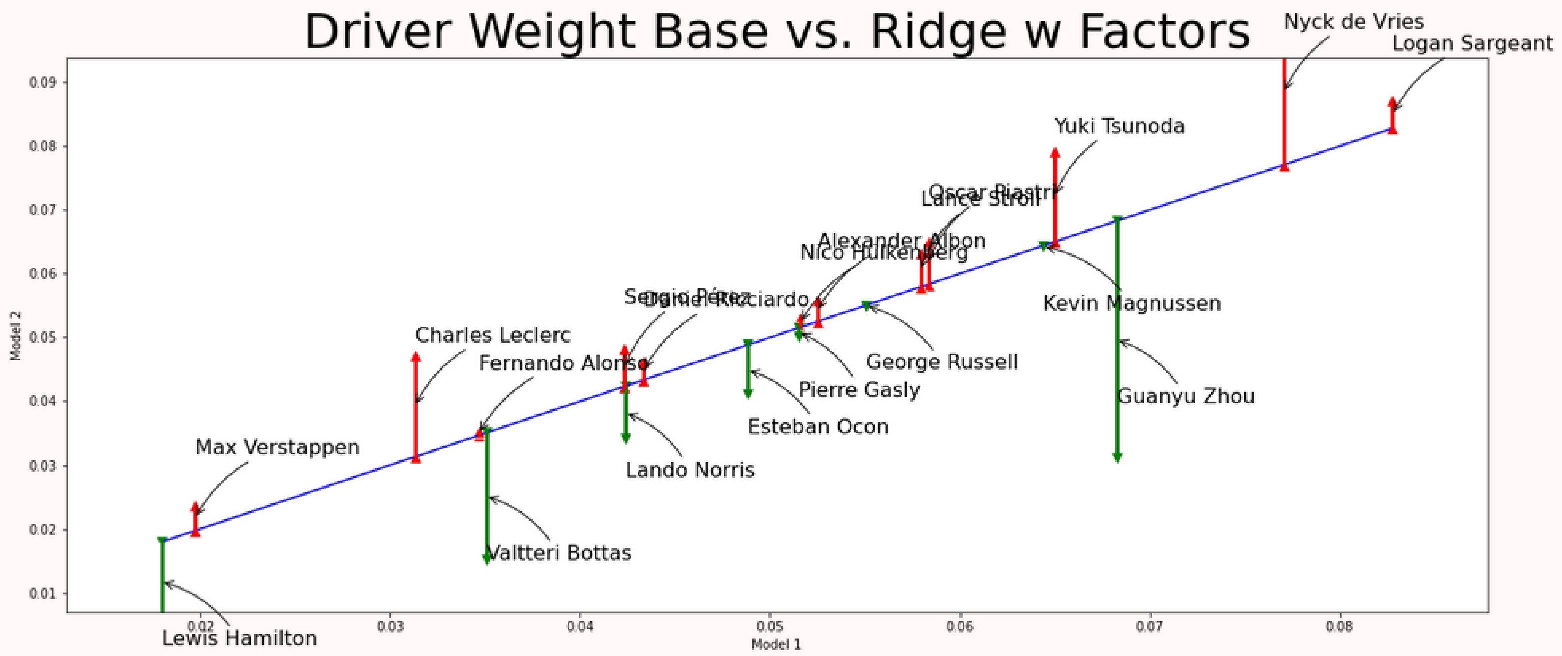


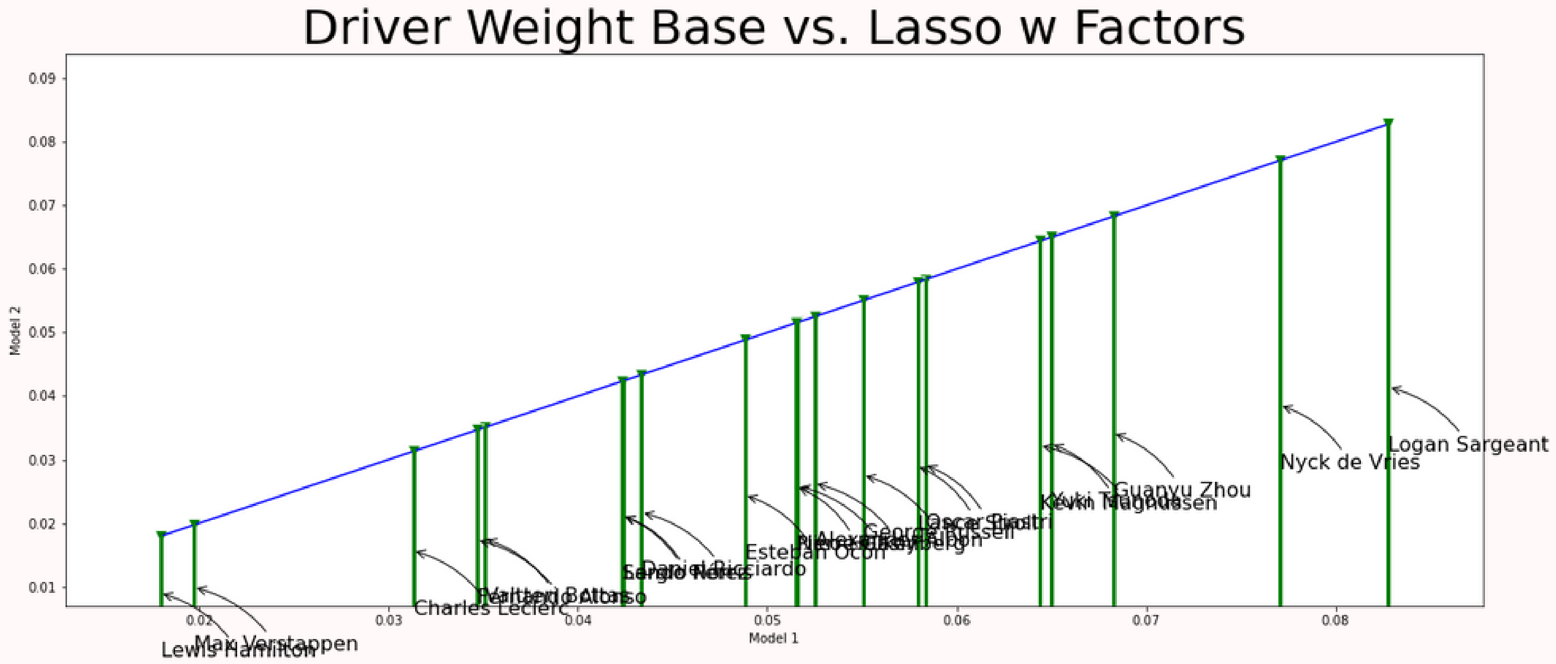
Imbalanced Data



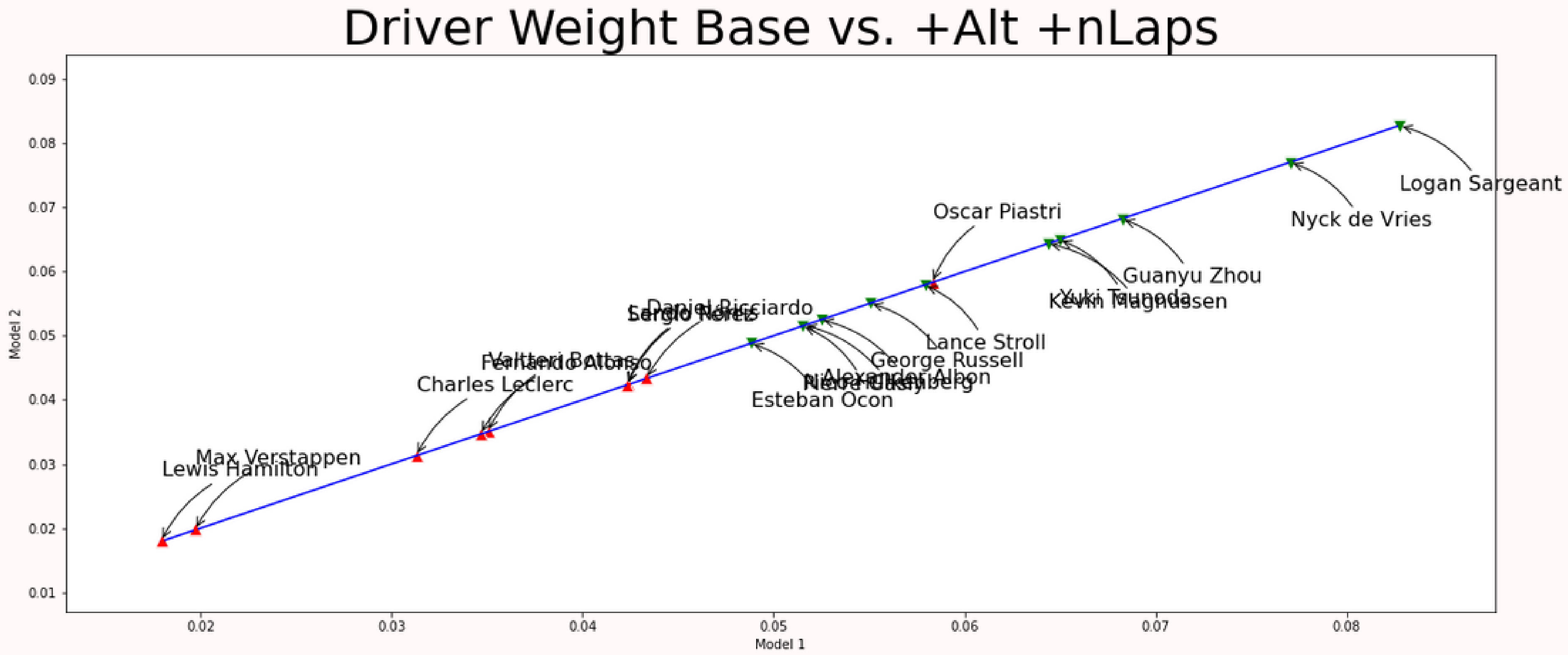
Adding Other Factors to Driver Model

Weights from Regularized Models



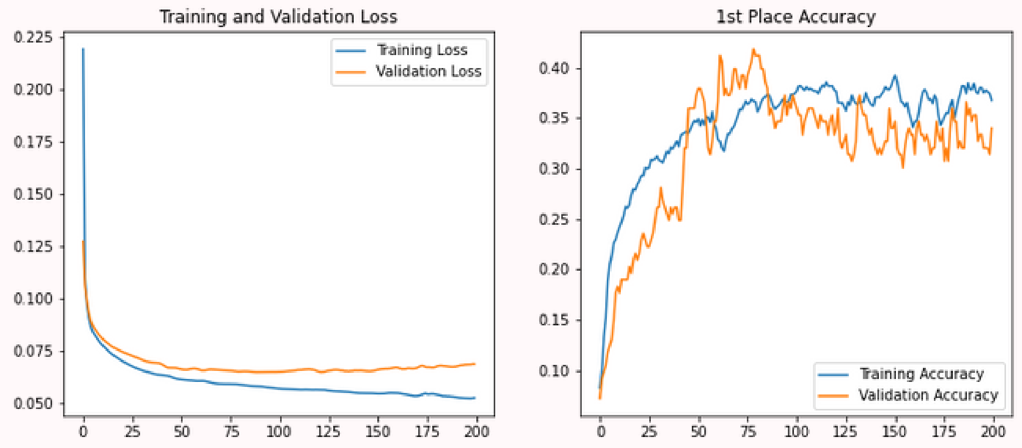


Driver Only vs. Chosen Model

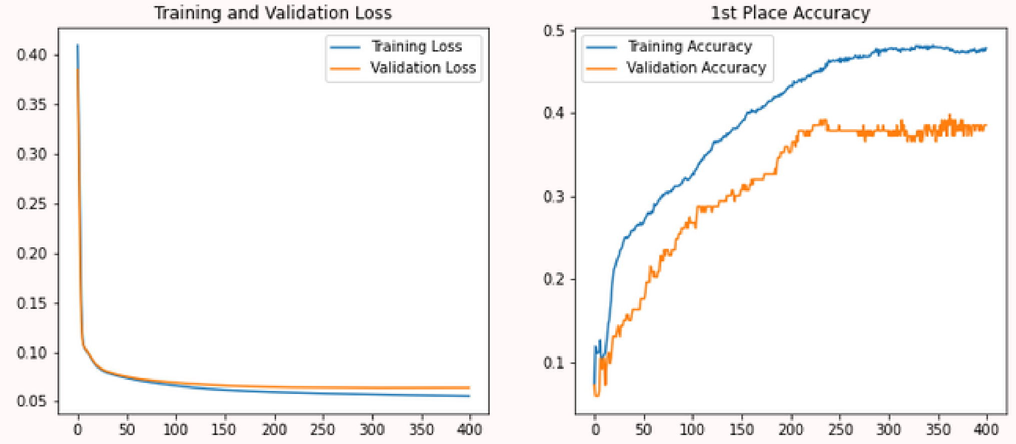


Results - Regression

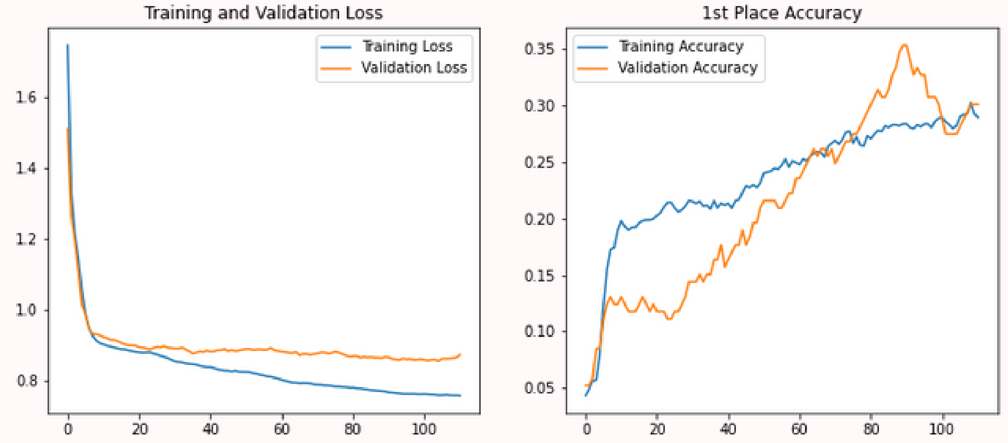
Laps 1-10, 1 Hidden Layer (MSE)



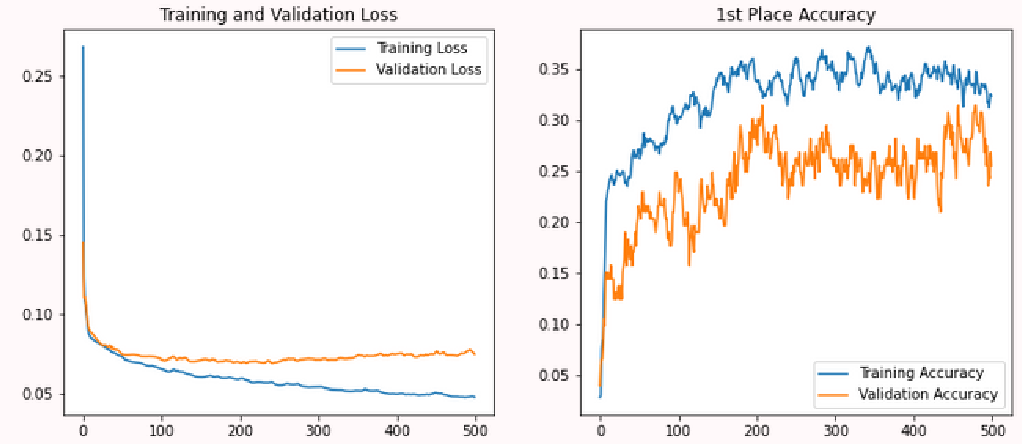
Laps 1-5 + Driver Coef, 1 Hidden Layer (MSE)



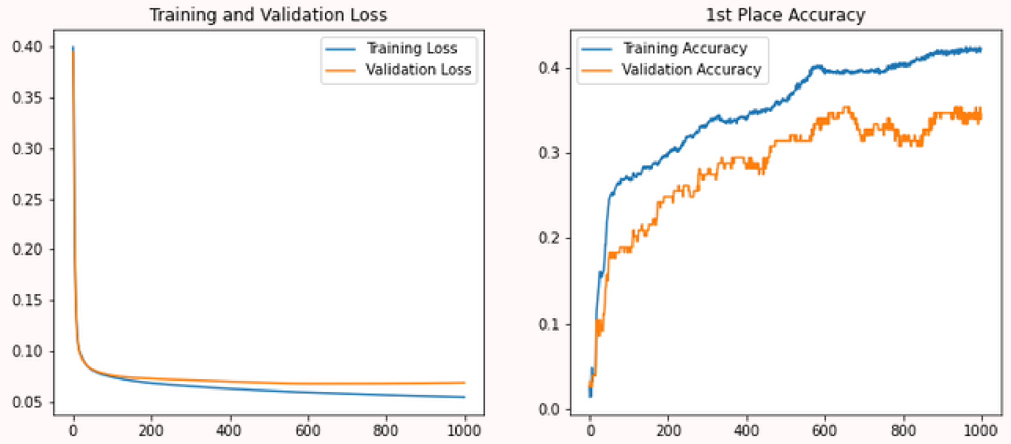
Laps 1-10, 1 Hidden Layer (custom ranking loss)



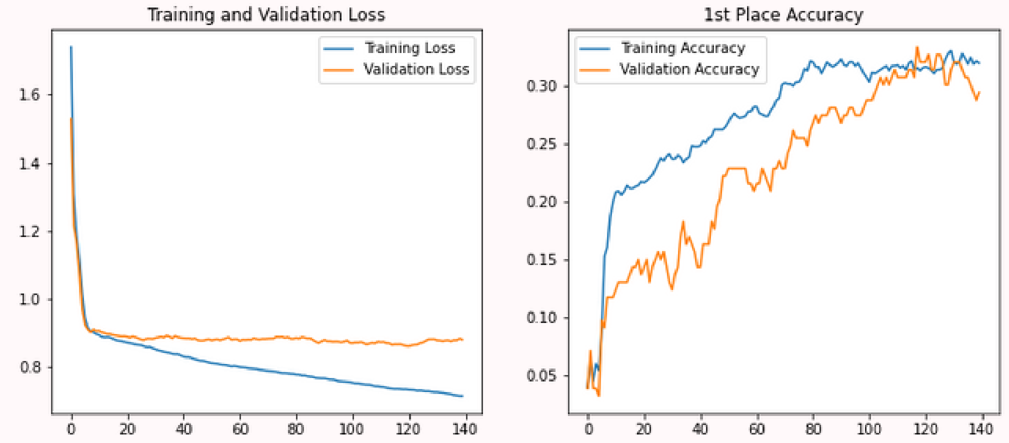
Laps 1-10, 2 Hidden Layers (MSE)



Laps 1-5 + Driver Coef, 2 Hidden Layers (MSE)

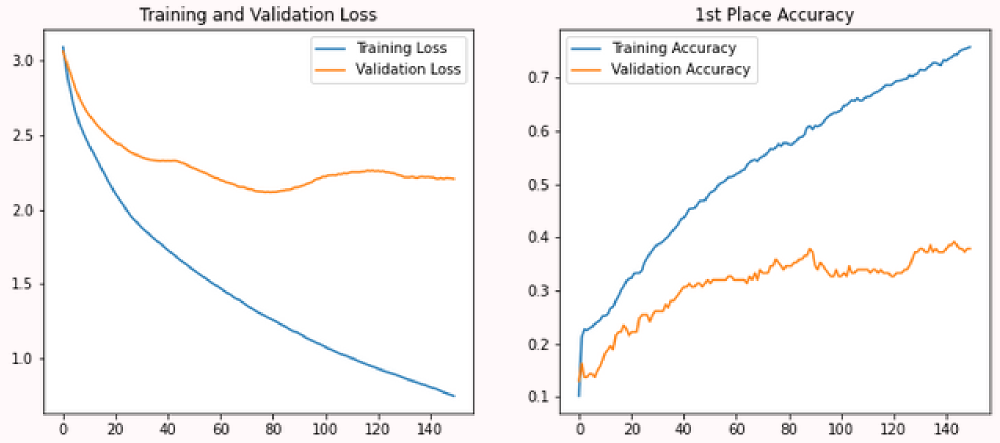


Laps 1-5 + Driver Coef, 2 Hidden Layers (custom ranking loss)

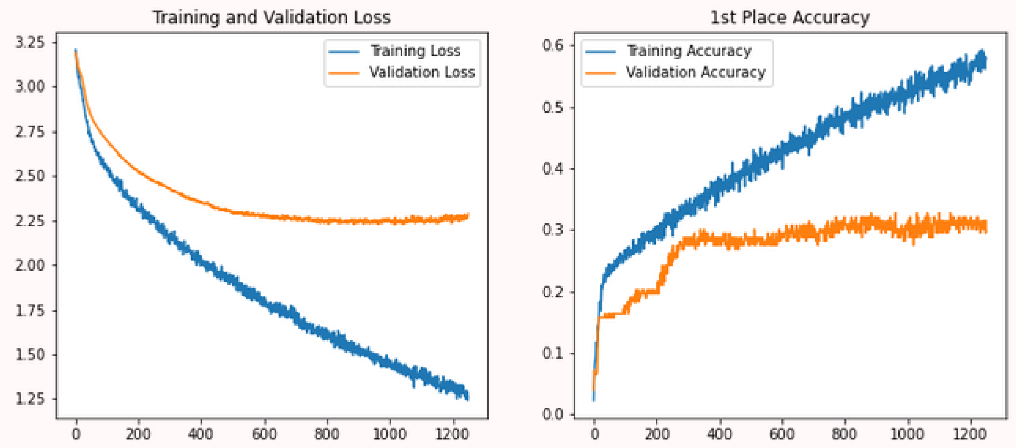


Results - Classification

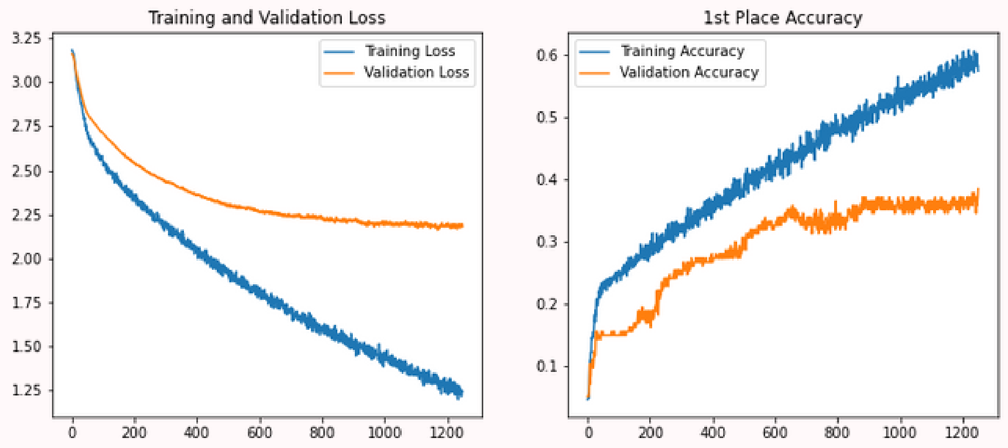
Laps 1-10, 1 Hidden Layer (CrossEntropy)



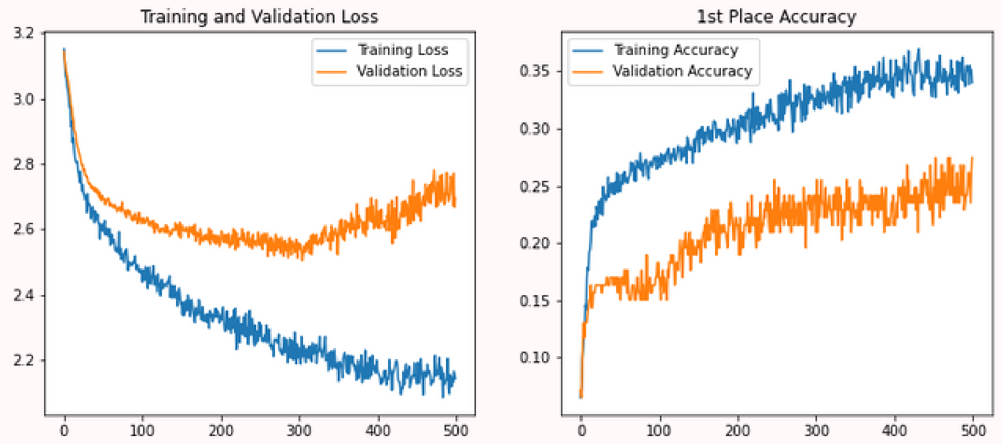
Laps 1-10, 2 Hidden Layers, dropout=0.2, lr= 0.0001 (CrossEntropy)



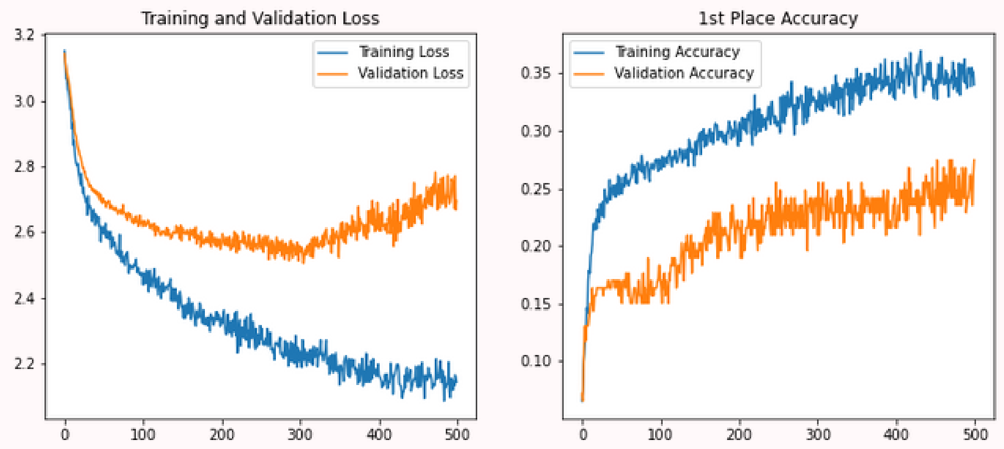
Laps 1-5 + Driver Coef, 2 Hidden Layers, dropout=0.2, lr=0.0001 (CrossEntropy)



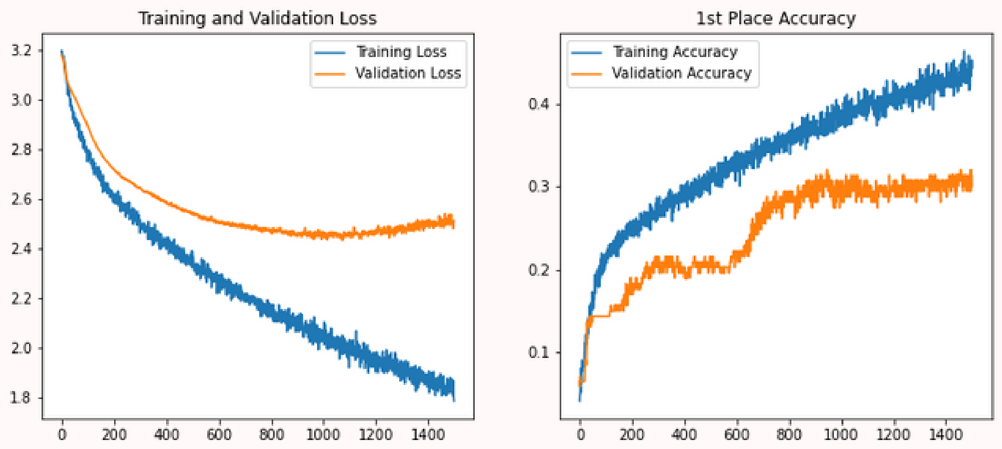
Laps 1-10, 2 Hidden Layers, dropout=0.5 (CrossEntropy)



Laps 1-10, 2 Hidden Layers, dropout=0.5, lr= 0.001 (CrossEntropy)



Laps 1-5 + Driver Coef, 2 Hidden Layers, dropout=0.5, lr=0.0001 (CrossEntropy)



Conclusions

- Regression models had higher first place accuracy than classification
- Classification models had high overfitting, dropout did not improve performance
- Best model: **Regression on Laps 1 to 5 + Driver Coef with 1 Hidden Layer (MSE) [epoch=363]**
    - Train 1st Place Accuracy: 0.478758
    - Valid 1st Place Accuracy: 0.392157
    - Test 1st Place Accuracy: 0.398693
- Limitations and weaknesses
    - The regression model has to predict if a driver will not finish the race, in addition to the ordering
    - The padded drivers may make the ratio of features to datapoints too large
    - Minor data leakage from driver feature model which was run on all races
    - Pitstops are not accounted for