# AI 539 Machine Learning Final Project

Bo Wang, Yen-Chun Chen

June 2024

## 1 Abstract

The final project includes "Gotham Cobs" which predicts duration based on multiple trip features and "Botanist" which can accurately predict plant leaf labels. Working on the "Gotham Cobs" problem was a fascinating experience, especially as we experimented with various regression models on a large training set. We implemented Linear Regression, Random Forest Regression, and Gradient Boosting Regression. Among these, the Gradient Boosting Regression model performed the best, achieving a Root Mean Square Error (RMSE) of 234.85 for duration prediction. In the Botanist project, we developed a Convolutional Neural Network (CNN) model to classify leaves into 38 different classes. The final CNN model, trained without data augmentation and with optimized parameters including four convolutional layers, 30 epochs, and a batch size of 64, achieved an accuracy of 98.43%, the second highest in the competition.

## 2 Project 1: Gotham Cabs

### 2.1 Introduction

This is a regression problem. We already have a trip record of different taxi routes in Gotham City, including the coordinates of passengers getting on and off, the time of passengers getting on and off, and the number of passengers. The machine learning model we designed needs to predict the duration of potential taxi trips in the future by learning the relationship between multiple variables, such as the number of passengers, the coordinates of getting on and off, and time information.

Here are the technologies considered for this prediction:

1. Considered regression algorithms:

- Linear Regression
- Random Forest Regression
- Gradient Boosting Regression (Final Choice)

2. Optimal hyperparameter combination selection approach:
   - Random Search Algorithm
3. Method for evaluating model performance:
   - Cross-validation

## 2.2 Optimal Hyperparameter Combination Selection Approach

For random forest and gradient boosting algorithms, there are a large number of hyperparameters that can be tested. If there is no specific estimate for these hyperparameters, it is very time-consuming to manually adjust the hyperparameters and gradually train them.

Grid search and random search are algorithms used to find the best model parameters. Grid search evaluates model performance by exhaustively trying all possible parameter combinations.

Such grid search is still too computationally expensive. In contrast, random search randomly samples hyperparameters within the value range to find relatively high-quality hyperparameter combinations. Therefore, I chose random search to determine the optimal parameter combination.

## 2.3 Method for evaluating model performance

Cross-validation can divide data into multiple folds to evaluate model performance. In this project, I use 5-fold cross-validation.

It can obtain a more robust model generalization performance evaluation, also the computational cost of 5-fold cross-validation is relatively low compared to 10-fold cross-validation.

## 2.4 Linear Regression

Linear regression is a very basic regression model used to predict the linear relationship between one or more independent variables and a dependent variable.

1. After training the linear regression model, through 5-fold cross validation of the linear regression model, we can get the $R^2$ **scores and MSE scores:**

Linear Regression Mean 5-fold Cross-Validation $R^2$ scores score: 0.60

Linear Regression Mean 5-fold Cross-Validation MSE score: 117703.87

2. We can see from its large Mean squared error and small $R^2$ score that it is not suitable for this problem. I think the reasons may be:

   - Linear regression has difficulty capturing this nonlinear relationship.
     The relationship between taxi travel duration and independent variables may be nonlinear. For example, during rush hour, travel duration may increase dramatically as the number of passengers increases.
   - High-dimensional data makes linear regression difficult to fit.
     There are too many independent variables. Linear regression is prone to overfitting on high-dimensional data, resulting in poor generalization ability.

## 2.5 Random Forest

Random Forest combines the predictions of multiple decision trees to make the final prediction.

1. Define the parameter distribution for random search, and find the best parameter combination in the following random forest parameter value range:

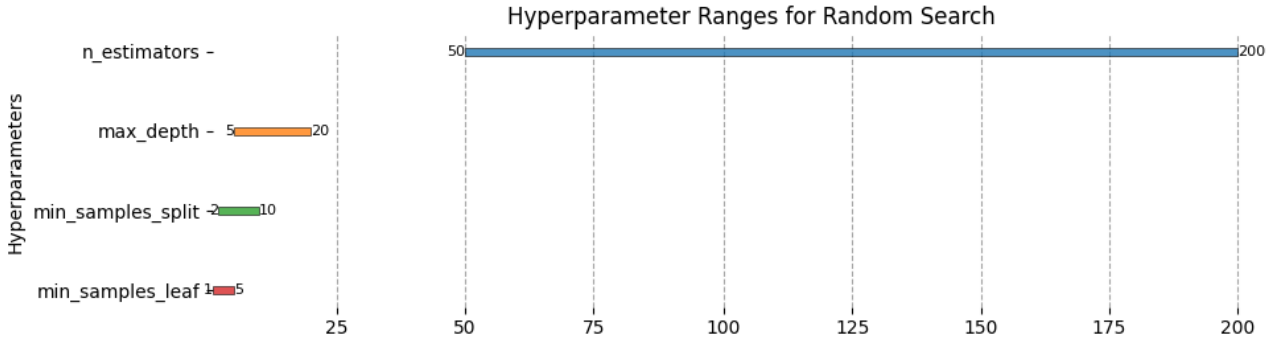


Figure 1: Hyperparameter ranges for Random Forest

The best parameter combination of random forest obtained is as follows:

| Parameter | Value |
|---|---|
| max_depth | 17 |
| min_samples_leaf | 4 |
| min_samples_split | 2 |
| n_estimators | 98 |

Figure 2: Table of optimal hyperparameter for Random Forest

2. Random forest model training using this set of parameter combinations, and performing 5-fold cross-validation on it.

   The $R^2$ **scores and MSE scores** we got:

   Random Forest Mean 5-fold Cross-Validation $R^2$ score: 0.78

   Random Forest Mean 5-fold Cross-Validation MSE score: 64371.40

   **Performance Analysis:** It can be seen that the performance of this random forest is much better than that of the linear regression model, but in order to achieve better prediction results, we can also try the Gradient Boosting Regression model.

## 2.6 Gradient Boosting Regression

Gradient Boosting Regression generates the final prediction by iteratively training a series of decision trees and summing up their predictions.

1. Define the parameter grid for random search, and get the optimal hyperparameter combination in the following value range:

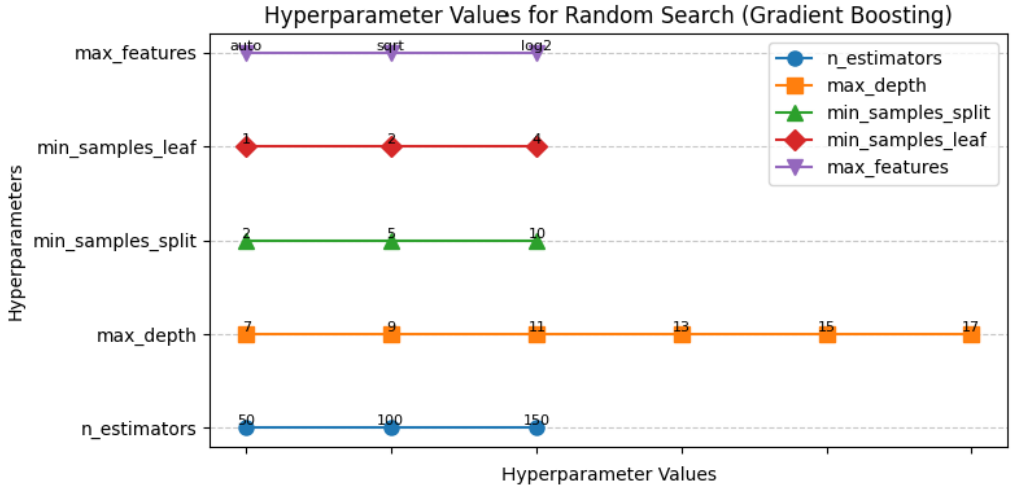


Figure 3: Hyperparameter ranges for Gradient Boosting

The best parameter combination is:

| Parameter | Value |
|---|---|
| n_estimators | 100 |
| min_samples_split | 5 |
| min_samples_leaf | 2 |
| max_features | sqrt |
| max_depth | 15 |

Figure 4: Table of optimal hyperparameter for Gradient Boosting

2. Use this set of parameter combinations to train the Gradient Boosting Regression model, and perform 5-fold cross-validation on it.

   **$R^2$ scores and MSE scores:**

   GBR Mean 5-fold Cross-Validation $R^2$ score: 0.81

   GBR Mean 5-fold Cross-Validation MSE score: 55852.85

3. **Performance Analysis:** We can see that the numerical performance of gradient boosting regression in MSE and $R^2$ is much better than that of linear regression, and it also has certain advantages over random forest.

   There are two advantages of Gradient Boosting Regression we found:

   - it can efficiently learn the non-linear relationship between features to prompt prediction ability.
   - it can control parameters to balance the accuracy and generalization ability of the model.

## 2.7 Concluding Remarks

Finally, I chose the Gradient Boosting regression model based on the excellent $R^2$ and MSE scores.

I use two histograms to visually compare the $R^2$ and MSE scores performance of these three sets of models.


Figure 5: $R^2$ Comparison (GBR has the biggest $R^2$)

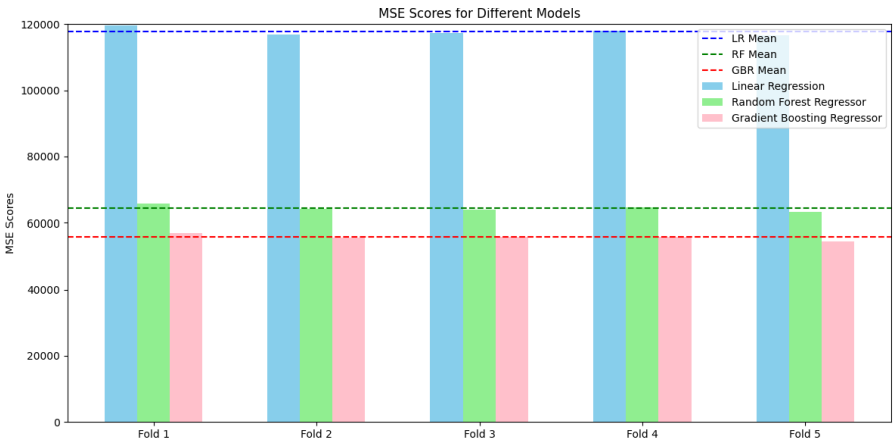


Figure 6: MSE Comparison (GBR has the smallest MSE)

# 3 Project 2: Botanist

## 3.1 Introduction

In this project, we need to develop a classification model to classify leaf images into 38 different classes representing various plant breeds and potential diseases. We utilize Convolutional Neural Networks (CNNs) because of their effectiveness in image classification problems. We used PyTorch instead of TensorFlow since Pytorch is more popular in the industry.

## 3.2 Model Structure

I designed a CNN model called **LeafCNN**:

- Four convolutional layers with increasing filter sizes (32 > 64 > 128 > 256).
- A ReLU activation function and a max-pooling layer follow each convolutional layer.
- A flatten layer to flatten a 3D tensor into 1D tensor.
- A fully connected layer with 512 units.
- Two Dropout layers with a dropout probability of 0.5 to prevent from overfitting.
- An output layer with 38 units (38 classes) for classification.

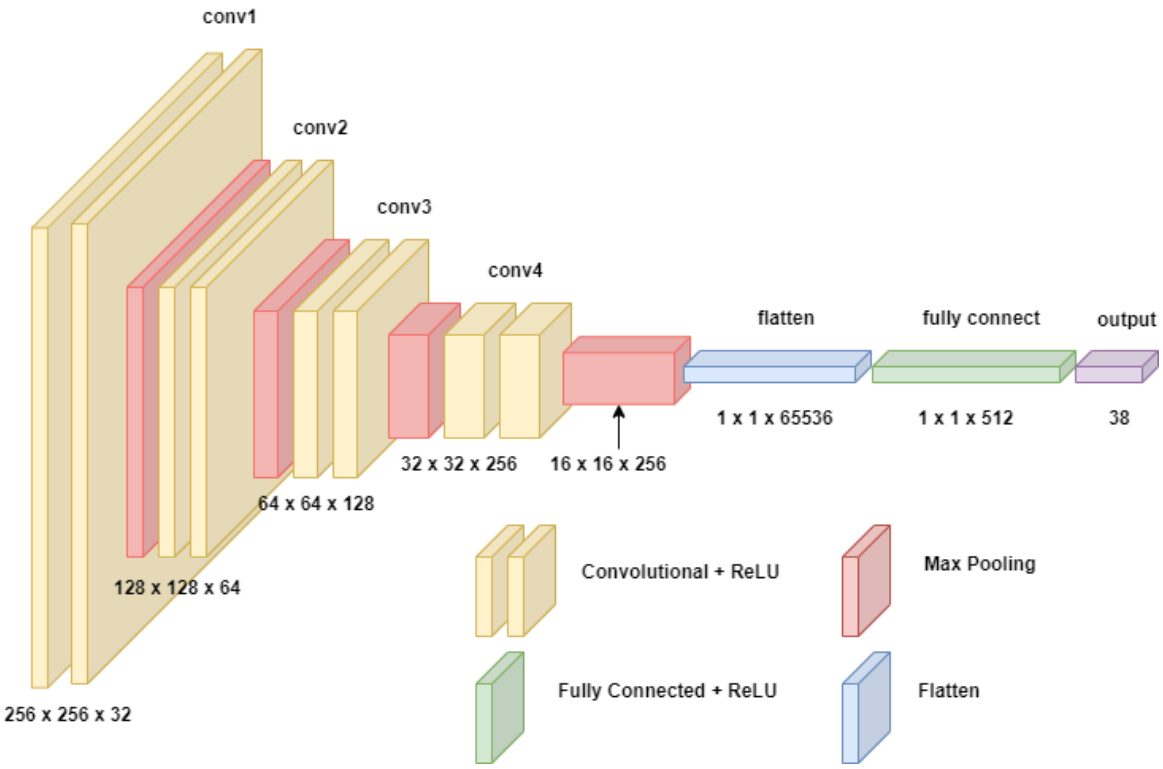


Figure 7: The archtiecture diagram of **LeafCNN**

## 3.3 Trainging Flow

The training dataset includes 50,000 images of leaves, labeled from 1 to 38, representing various plant breeds and health conditions. We used PyTorch's Adam optimizer with a learning rate of 0.001 and the cross-entropy loss function for the training process. This setup aims to achieve high accuracy in classifying the different plant types and their health states.

### 3.3.1 Data Processing

I split the overall dataset into 45,000 training images and 5,000 testing images. To increase the diversity of the training data, I applied data augmentation techniques such as flipping and rotating the images. These augmentations aim to improve the model's ability to generalize by providing a wider range of variations. After augmenting the data, I converted the images into tensors, making them compatible with the neural network's input requirements. This preparation ensures that the data is ready for efficient training and testing.

### 3.3.2 Training Loop

Within each epoch, I trained the model on 45,000 training images and evaluated it on 5,000 validation images. First, I cleared the previous gradients. During the forward pass, the input data passed through the neural network to generate predictions. Next, the loss function measured the accuracy of these predictions against the true labels. In the backward pass, gradients of the loss with respect to the model parameters were computed. Finally, the optimizer updated the model parameters using these gradients.

## 3.4 Parameter Optimization

Initially, I designed the model with only two convolutional layers, set the number of epochs to 20, and the batch size to 32, which resulted in an accuracy of approximately 91%. To optimize the model and improve accuracy, I experimented with various parameters. These parameters included the number of convolutional layers, the number of epochs, batch size, and the use of data augmentation.

### 3.4.1 Number of Convolutional Layers

Test different number of layers as the only variable to calculate loss and accuracy. We can see the 4-layer one has the best accuracy and the lowest cross-entropy loss.

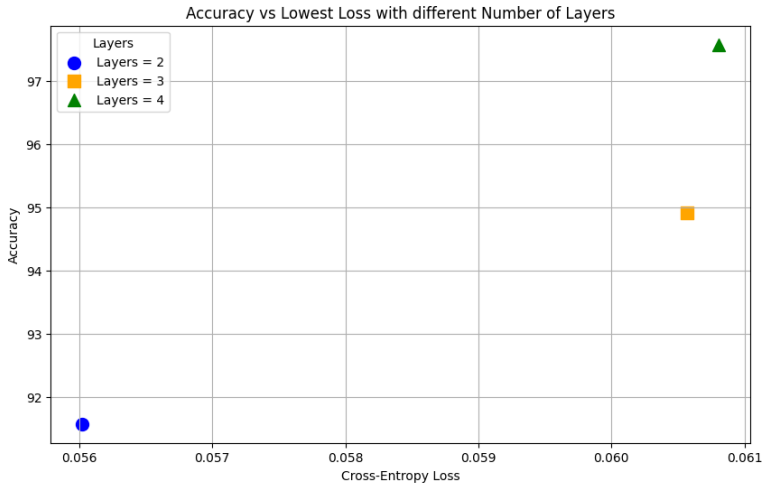


Figure 8: The comparison of different number of convolutional layers.

### 3.4.2 Number of Epoch

Test different number of epochs as the only variable to calculate loss and accuracy. We can see the epoch number of 30 has the best accuracy, however, the epoch number of 20 has the lowest loss. The

reason might be too overfitting to generalize as effectively to new data. In addition, cross-entropy loss and accuracy different aspects of performance.Loss measures the difference between the predicted probabilities and the actual labels, while accuracy measures the proportion of correct predictions. It is possible for the model to have a very confident wrong prediction that lowers the loss significantly but doesn't improve accuracy.

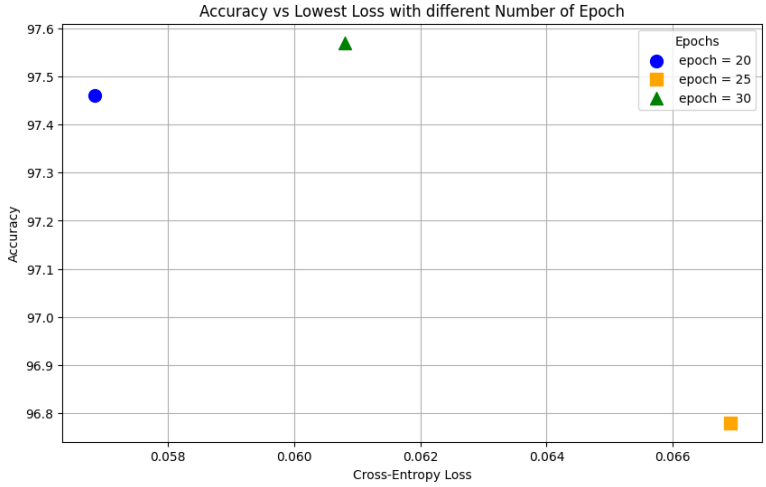


Figure 9: The comparison of different number of epochs.

### 3.4.3 Batch Size

Test different batch sizes as the only variable to calculate loss and accuracy. We can see the batch size 64 has the best accuracy and the lowest cross-entropy loss.

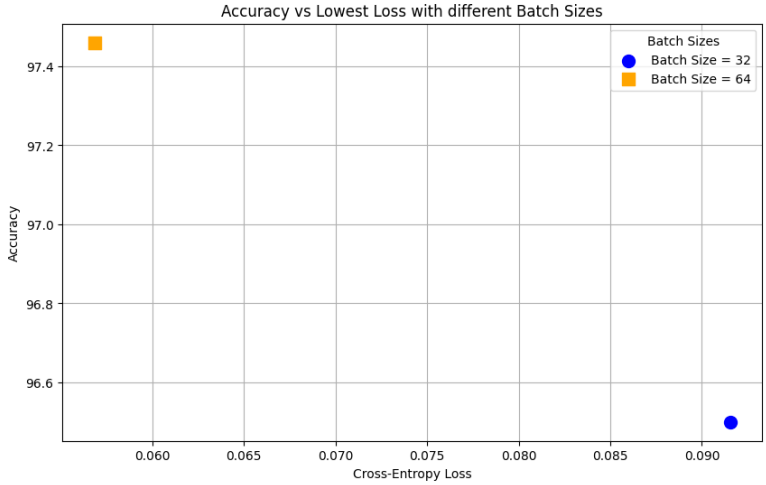


Figure 10: The comparison of different batch sizes.

### 3.4.4 Data Augmentation

Test different batch sizes as the only variable to calculate loss and accuracy. We can see the one without data augmentation has the best accuracy and the lowest cross-entropy loss. This is quite surprising since generally more diverse data is better for a model. We got the result because the original dataset might already be diverse and representative enough, thus, reducing the need for augmentation.

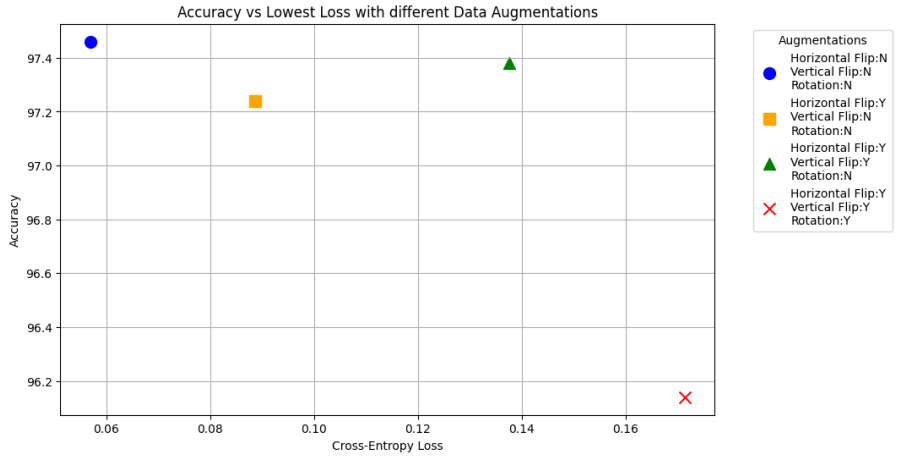


Figure 11: The comparison of different combinations of data augmentation.

## 3.5 Summary

Ultimately, I found that the CNN model with four convolutional layers, trained for 30 epochs with a batch size of 64, and without data augmentation, achieved the highest prediction accuracy. This configuration outperformed other setups, indicating that the dataset's inherent quality and diversity were sufficient without additional augmentation. With these fine-tuned parameters, I used all 50,000 images in the dataset as the training set to train the final model. The final model achieved an accuracy of 98.43%, which is the highest accuracy I have ever received. This result shows the importance of careful hyperparameter tuning and the effectiveness of the chosen model architecture for this specific task, and the process confirmed the value of iterative experimentation and validation in developing high-performing neural network models.