User/Developer Manual

VehID

Version 1.0

Remington Greko, Spencer Hirsch, Thomas Johnson, Alexis Nagle

Project Advisor: Dr. Marius Silaghi
Project Client: Clayton Levins, Executive Director of Smart North Florida

April 10, 2024

Contents

1 Introduction

1.1 Purpose

This User/Developer manual describes the system features as well as system design of VehID, vehicle identification software. This system was created with law enforcement in mind, the objective was to create a system that could be used by both local and federal law enforcement to assist in vehicle related crimes such as AMBER Alerts.

1.2 Scope

Our software, VehID, is a machine learning model that utilizes a series of six neural networks (NN) to accurately identify vehicles based on set criteria. Each neural network has a specific task that contributes to the overall system. The overall system has three main components, vehicle extraction, identifying features, and license plate detection and extraction. Under these components we have a varying number of neural networks that collaborate to complete these tasks. Vehicle extraction has one NN responsible for identifying vehicles in a frame. The identifying features component has three separate NNs: identifying color, identifying vehicle make, and identifying the body-type of the vehicle. Lastly, the license plate detection and extraction component consists of two NNs: identifying a license plate on the vehicle and a separate model that identifies that characters on the license plate. In the event that a match is found location of the the vehicle is reported to the proper authorities. This system would automate the entire process of locating vehicles found to be involved in crimes.

1.3 Overview

This document is intended to aid users and future developers in understanding how to utilize our system both through the front-end web application as well as utilizing the source files to modify and run the system. The following sections will allow the user to understand all available functionalities and have detailed explainations of how to work each component. It will also allow any future devopers to understand the system architecture as well as provide detailed descriptions of all source files utilized in the system.

2 User Manual

2.1 System Features

2.1.1 Create Queries

One of the main ways a use can interact with our system is to create search queries. These search queries allow for the user to check for a specified vehicle against the database of identified vehicles. To create a query a user must first navigate to the page to input a query by selecting the 'Input a Query' tab at the top of the web application.

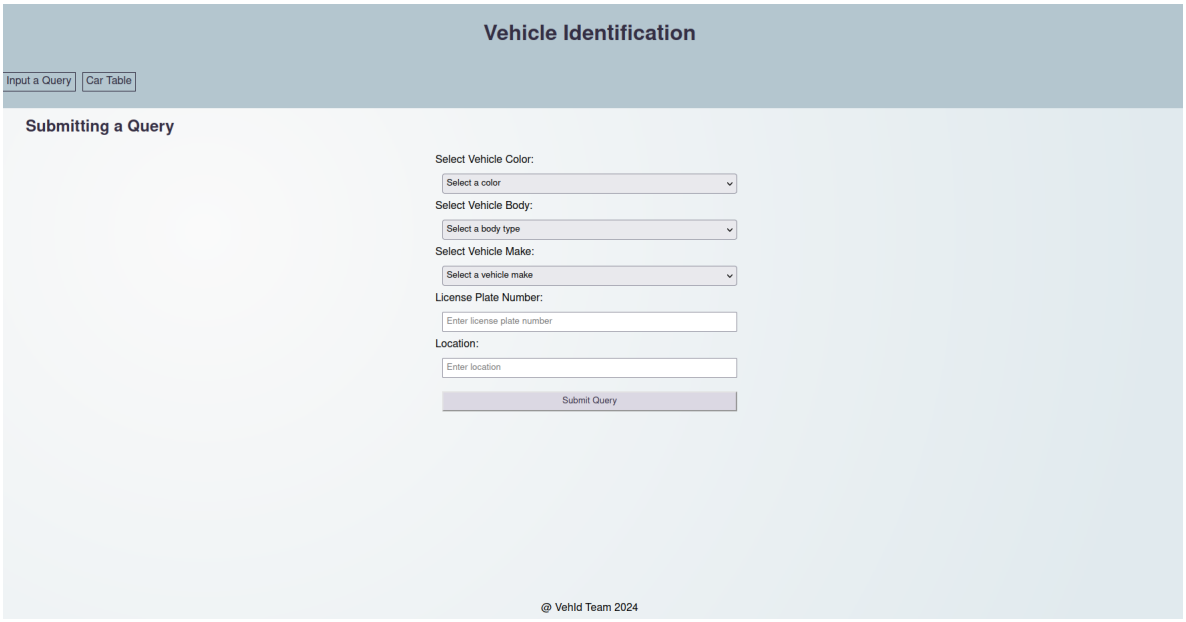

Figure 1: Input a Query page on web application.

Once the user has navigated to this page, they can fill in the desired fields to specify their search. To submit the query, the user will then select the 'Submit Query' button at the bottom of the screen. A modal will display in the center of the screen showing all full and partial matches along with any captured image of the vehicle. This modal may be closed to display the original screen.

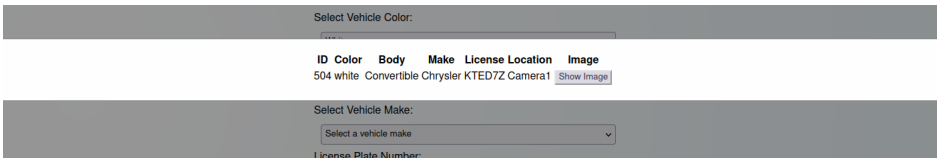

Figure 2: Image of query output without image.

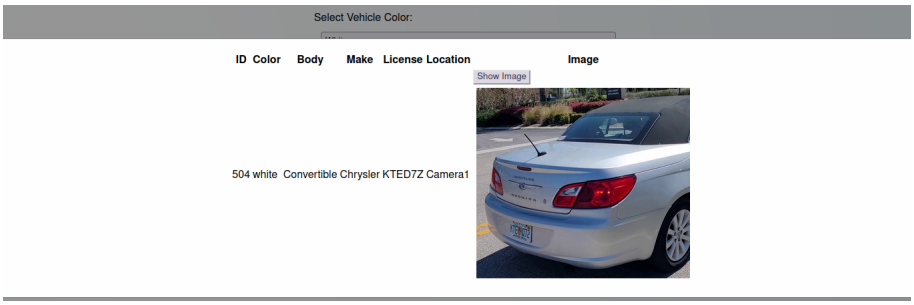

Figure 3: Image of query output with image.

2.1.2 View Vehicle Database Table

Users have the ability to view all identified vehicles contained in the database. To do so, a user will navigate to the vehicle page by selecting the 'Car Table' tab at the top of the page.

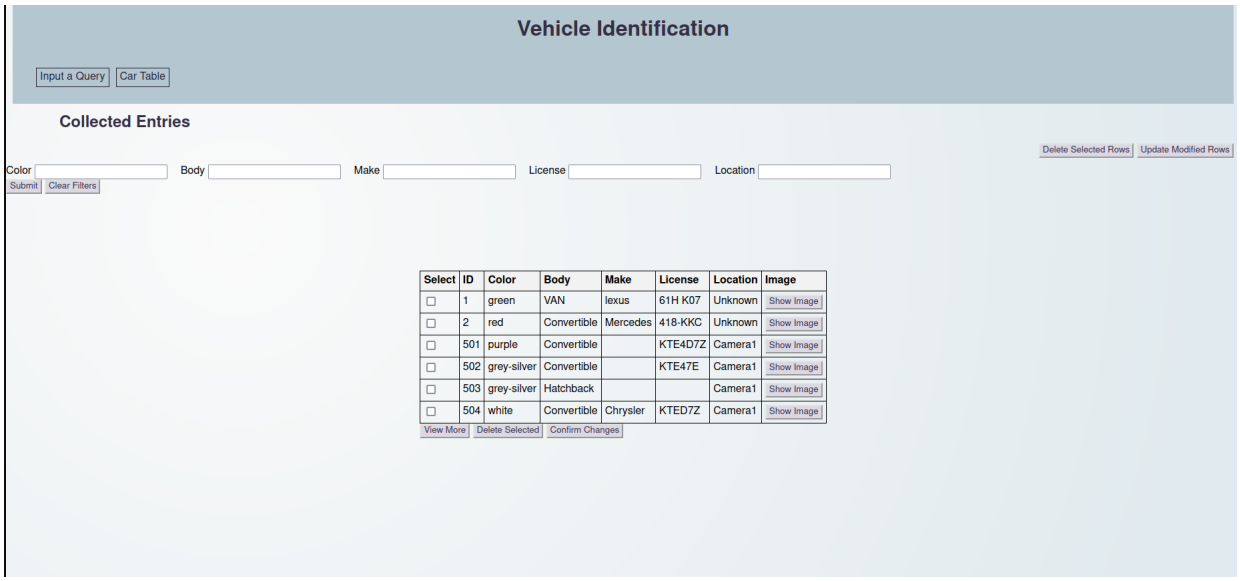

Figure 4: Image displaying 'Car Table' data table.

2.1.3 View Vehicle Entry Image

For manual verification, a user is able to view an image of the detected vehicles included in the vehicle database. To perform this action, the user must first navigate to the vehicle page by selecting the 'Car Table' tab. Once on the vehicle page, the user will identify the vehicle entry they would like to examine and select the 'Show Image' button to view an image of the detected vehicle.

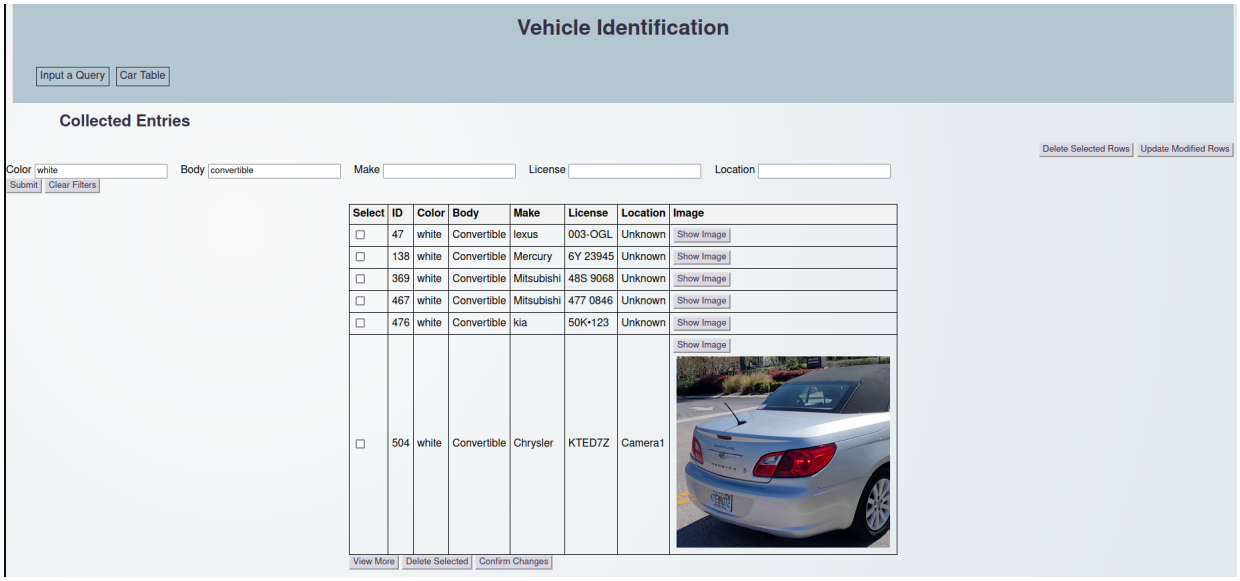

Figure 5: Image displaying vehicle information in 'Car Table' data table.

2.1.4 Filter Vehicle Database Entries

On the 'Car Table' page, a user is able to filter entries based on the criteria they wish to view. They have a variety of options: Color, Body, Make, License, and Location. A user may choose which information they would like to fill in, allowing them to search our database for a partial match. In addition to these input options, this portion also consists of a 'Submit' and 'Clear Filters' button. Once a user inputs the information they would like to search for they can select submit and it will display all matches found in the database.

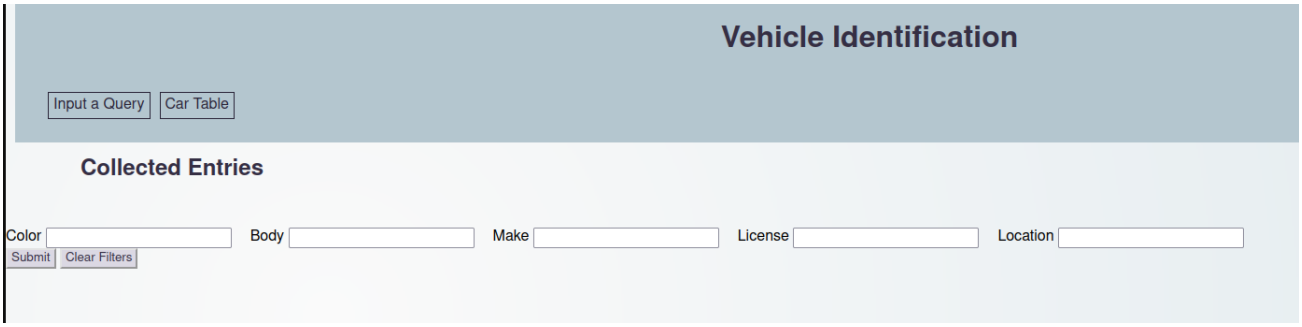

Figure 6: Image displaying the components of the filter function.

2.1.5 Remove Vehicle Database Entries

Users have the ability to manually remove entries from the database of identified vehicles in the case that the entry is not valid or not necessary. To do so the user must first navigate to the vehicle page by selecting the 'Car Table' tab. Once on the vehicle page, the user will identify any vehicle entries they would like to remove and click the check box located in the

'Select' column for that entry. Once selecting the check box, the column will be highlighted red to show the user which rows they have selected.

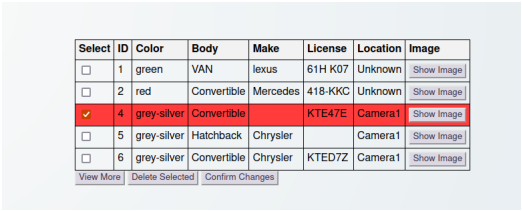

Figure 7: Image of row selected.

Once the user selects all the entries they would like to remove, the user will then select the 'Delete Selected' button located below the table. After selecting the 'Delete Selected' button, a pop-up will appear at the top of the screen verifying the user would like to delete the selected rows. To proceed with the action, the user must select 'OK'. To cancel the action, the user must select 'Cancel'.

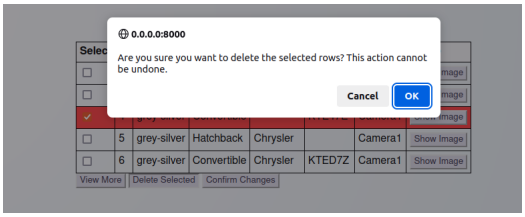

Figure 8: Image of deletion verification.

If the user proceed with the action, the selected rows will be deleted and a pop-up will appear at the top of the screen confirming that the selected rows have successfully been deleted. The user must select 'OK' to dismiss the message.

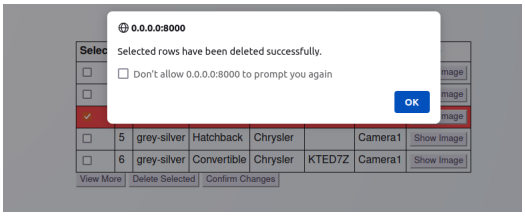

Figure 9: Image of successful deletion.

2.1.6 Edit Vehicle Database Entries

Users have the ability to manually edit entries from the database of identified vehicles in the case that the entry is not correct or missing information. To do so the user must first navigate to the vehicle page by selecting the 'Car Table' tab. Once on the vehicle page, the user will identify any vehicle entries they would like to edit. The user will then select the fields of the entries they would like to modify by simply clicking on them and typing directly

into the fields. Fields the user is typing into will display a red outline. Once the user has made all the modifications that they want, they will select the 'Confirm Changes' button located below the table.

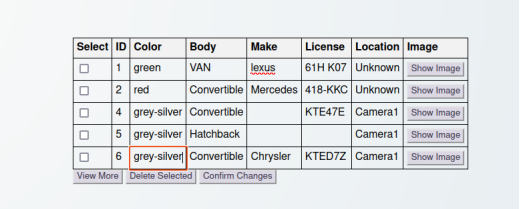

Figure 10: Image showing user changing field in table.

After selecting the 'Confirm Changes' button, a pop-up will appear at the top of the screen verifying the user would like to confirm the modifications made. To proceed with the action, the user must select 'OK'. To cancel the action, the user must select 'Cancel'.

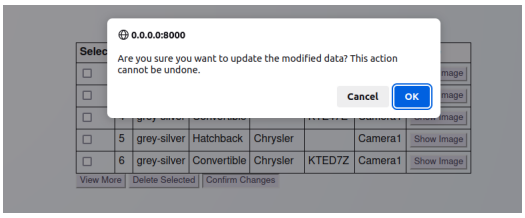

Figure 11: Image showing a pop-up confirming if the user wants to update modified data..

If the user proceed with the action, the modifications will be saved to the database and a pop-up will appear at the top of the screen confirming that the selected rows have successfully been deleted. The user must select 'OK' to dismiss the message.

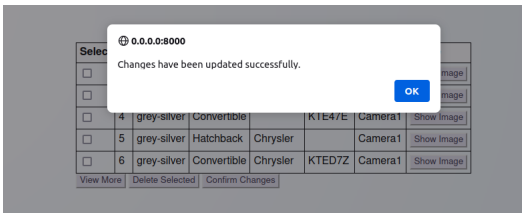

Figure 12: Image showing the pop up confirming the changes were updated..

3 Developer Manual

3.1 System Architecture

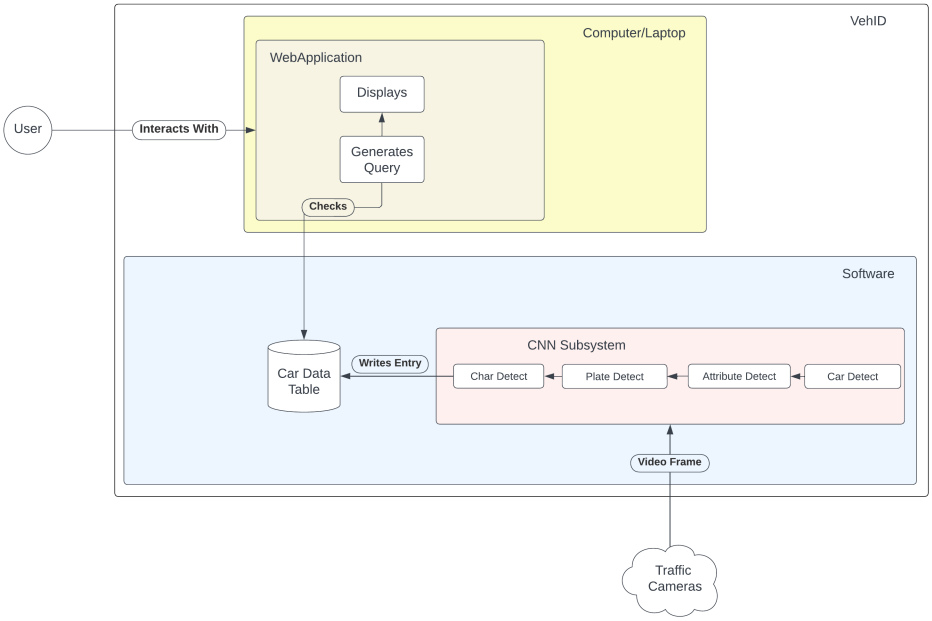

Figure 13: VehID System Architecture

The system is compromised of three primary portions: a neural network subsystem, database, and web application.

3.1.1 Neural Network Subsystem

The Neural Network subsystem as a whole takes in video footage and returns a JSON file of all vehicles detected within the video as well as the characteristics predicted from the series of neural networks. The subsystem architecture can be seen below:

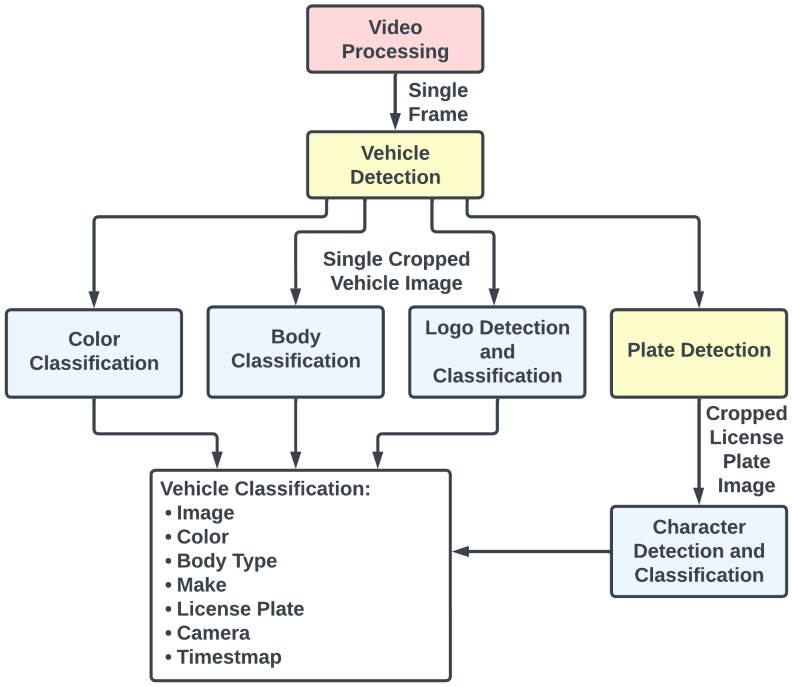

Figure 14: Neural Network Subsystem Architecture

The subsystem begins with a video processing portion of code. In this portion, the Python package OpenCV is utilized to open the video footage and step through the video at a specified rate to extract individual frames. These individual frames are then passed into the vehicle detection model.

The vehicle detection model is a Ultralytics YOLOv8n trained with 30 epochs with a early stopping patience of 7. This performed fairly well resulting in a 95% training accuracy. The testing accuracy was slightly lower at 75% accuracy. Due to these lower results, we restricted returned predictions to those with a confidence level higher than 95%.

Once a prediction has been made, an more precise image of the vehicle is extracted and then processed through the following models we have constructed: color classification, body type classification, logo detection and classification, and license plate detection.

The first two models, color and body type classification, both are built from a Mini VGGNet architecture. This architecture consists of four convolutional layers, six activation layers, two dense layers, and two max pooling layers.

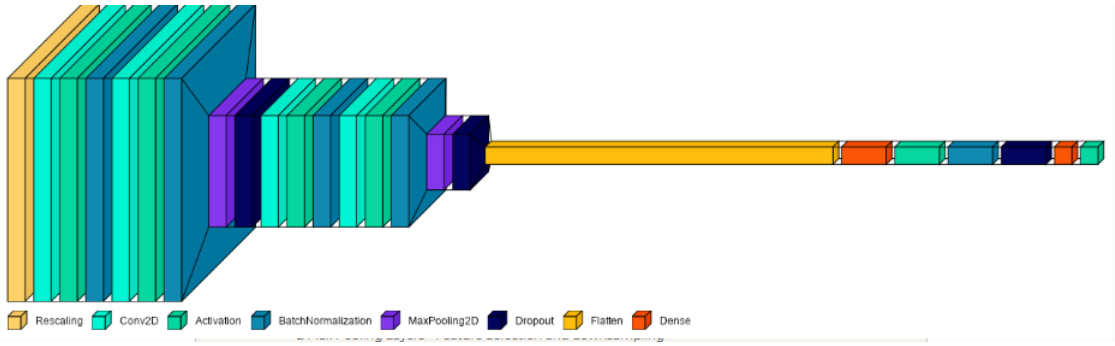

Figure 15: Mini VGGNet Architecture

The color classification model was trained with an SGD optimizer and incorporation of L1 and L2 regularization, random weight initialization, and reducing learning rate upon plateau. With these attributes, the model was able to achieve a testing accuracy of 91%. Upon classifying the image, the resulting classification and confidence level are stored into a dictionary.

The body type classification model was trained with an Adam optimizer and incorporation of L1 and L2 regularization, random weight initialization, and reducing learning rate upon plateau. With these attributes and restoring the best weights found, the model was able to achieve a testing accuracy of 84%. Upon determining the body type of the given vehicle, the resulting classification and confidence level are stored into the dictionary of characteristics for the vehicle.

The logo detection and classification model is an Ultralytics YOLOv8m trained with 30 epochs and an early stopping patience of 5. This model performed with a 75% training accuracy and a 70% testing accuracy. Upon detecting and classifying the logo in the image of the given vehicle, the resulting classification and confidence level are stored added to the dictionary of vehicle characteristics.

The license plate detection model is an Ultralytics YOLOv8n trained with 30 epochs and an early stopping patience of 5. This model performed well with both a training and testing accuracy of 95%. Upon detecting the vehicles license plate, a cropped image of the vehicles license plate is extracted to use in the license plate character detection model.

The license plate character detection model is an Ultralytics YOLOv8s trained with 10 epochs and an early stopping patience of 3. This model performed well with a training and testing accuracy of 99%. Upon detecting and classifying any characters within the license plate image, the characters are sorted based upon the x coordinate of the bounding boxed to form a series of character which we determine to repersent the license plate number. This number as well as the corresponding confidence level is stored into the dictionary with the other information from this vehicle.

Once the extracted vehicle image is passed through all the neural network models, the dictionary with the corresponding characteristics as well as the vehicle camera, timestamp, and frame information are added to the JSON file which will continue to be appended to until all vehicles within all frames of the video are processed. The JSON file is then passed off to be utilize within the database.

3.1.2 Database

The data output from the CNN subsystem was in JSON format so we based our choice of data storage off this fact. To allow for easy development and direct storage of the JSON we chose JSONBin.io. This is a simple data storage service with a free tier allowing us sufficient storage for the output of the CNN subsystem. We researched other services such as AWS, Google Cloud, Microsoft Azure, and others, but we decided that it would be too time consuming to use these services especially for the basic data format we had.

3.1.3 Web Application

The web application was made using native html, css, javascript, and node. Web pages have a uniform header, body, and footer style kept in main.css. index.html page containing the query submit form uses template literals to fill out each applicable drop down. Once field(s) are filled in, the submit buttons event listeners function handleSubmit will be run upon clicking submit. This will create a resultsObjs for each eventFilter. Once the resultObjs is created the table is created using template literals, and displayed in a modal container. This modal will display in the center of the screen and may be closed to return the screen to its ooriginal state.

On the second page, car_table.html will display a tabulated view of the database. The pages body will contain text inputs stored in a filter class, and a div with the id of generated-table to be used by template literals. carLiterals.js will fetch the data and store it int othe variable ‘carList‘. Once fetched the entries are generated via the function ‘generateEntriesIncrement‘, displaying the first 20 entries, and check box event listeners are added to each table row. When the table is being created using template literals event litsteners are added to the ‘Show Image‘ buttons in the table rows with the function to display an image of the vehicle when clicked. The ‘Delete Selected / Delete Selected Rows‘ buttons and ‘Confirm Changes / Update Modified Rows‘ buttons also have event listeners which will call functions to run our CRUD operations listed below

We designed the web application around the predetermined data format which would be outputted from the final version of the CNN subsytem. This meant that we simply had to allow users to access that data and interact with that. The overarching goal of the project was to allow users to interact with the CNN subsystem through a web application. Due to this goal, we decided an application which used CRUD operations to allow users to interact with the data would be the best choice. The data produced by the CNN subsystem was simply stored in JSON format, fetched, and tabulated to display to users. We chose this approach because it allows users to interact with a simple interface and perform actions without difficulty. Another thing we took into account was confirming modifications to the data, which we included on the tables to ensure users would not unintentionally delete or modify entries.

3.1.4 Neural Network Subsystem

The source files necessary for utilizing the neural network subsystem are the following:

- VideoProcessing.py: This file can be edited to process video files at specified file paths. This file dumps to a JSON file called feed.json
- color_rec_model.h5: This file contains the trained color recognition model needed to make vehicle color predictions.
- body_type128_model.h5: This file contains the trained body type recognition model needed to make vehicle body type predictions.
- logo_det_model.pt: This file contains the trained make recognition model needed to make vehicle make predictions.
- veh_det_model.pt: This file contains the trained vehicle detection model needed to detect vehicles in a frame.
- plate_det_model.pt: This file contains the trained plate detection model needed to detect license plate in an image.
- char_det_model.pt: This file contains the trained character recognition model needed to make vehicle license plate predictions.

3.1.5 Web Application/Database Subsystem

- index.html: This file contains the main page of the web application, with a form for submitting queries.
- car_table.html: This file displays a tabulated version of the car data, along with text fields for filtering, and buttons to edit and delete entries from the table.
- queries.html: This file displays a tabulated version of the query data (this was omitted from the final version of the project)
- main.css: This file contains the main css, and color schemes for the css.
- tables.css: This file contains the styling for tables.
- modal.css: This file contains the styling to display the modal in index.html.
- index.js: This file contains the code for handling form submissions for queries, and for displaying the modal.
- carLiterals.js: This file contains the code for all event listeners on car_table.html, and displays the data using template literals.
- fetchData.js: This file contains a function to test GET requests made to the JSONBin database.
- jbinAPI.kjs: This file contains functions to fetch, save, add, update, and delete records from the car database.

- jbinAPIQueries.js: This file contains functions to fetch, save, add, update, and delete records from the query database.
- networksToJbin.js: This file contains functions to fetch data from the database, add new records extracted from the CNNs, and save that updated data back to the database.
- queryLiterals.js: This file contains the code for all event listeners on query_table.html, and displays the query data using template literals (this was omitted from the final version of the project)

3.2 Description of Methods/Functions

3.2.1 Neural Network Subsystem

The file used for the neural network subsystem, VideoProcessing.py, has the following functions:

- vehicle_prediction: This function takes in an image of the road and utilizes the trained vehicle detection model to return the bounding boxes of all vehicles detected within the frame.
- color_prediction: This function takes in a cropped image of a vehicle and utilizes the trained color prediction model to return the color prediction and corresponding confidence level.
- body_prediction: This function takes in a cropped image of a vehicle and utilizes the trained body type prediction model to return the body type prediction and corresponding confidence level.
- make_prediction: This function takes in a cropped image of a vehicle and utilizes the trained make prediction model to return the make prediction and corresponding confidence level.
- plate_prediction: This function takes in a cropped image of a vehicle and utilizes the trained plate detection model to return the bounding boxes of the vehicle's license plate detected within the frame.
- char_prediction: This function takes in a cropped image of a vehicle's license plate and utilizes the trained plate character prediction model to return the license plate prediction and corresponding confidence level.
- crop_image: This function takes in both an image and bounding box coordinates and returns a cropped portion of the given image based upon the specified bounding boxes.
- predictFrame: This function takes in a single frame image, the frame number, camera name, and timestamp of the frame given. This function calls the functions above to gather predictions for every vehicle within the given frame and dumps all the predictions into a JSON file.

- processVideo: This function takes in the path to a video, the specified camera name the video was taken on, and the time the video was taken. This function opens the specified video and extracts frames at a given rate. These frames are then passed into the predictFrame function.

- main: The main function loads all the pre-trained models that are necessary to make predictions. This function calls the process video function on any specified file path for the videos that need to be processed.

3.2.2 Web Application/Database Subsystem

- generateColorOptions(): This function generates HTML option elements for a selection dropdown menu with color options.

- generateBodyOptions(): This function generates HTML option elements for a selection dropdown menu with body options.

- generateMakeOptions(): This function generates HTML option elements for a selection dropdown menu with make options.

- generateQueryLocations(): This function generates HTML option elements for a selection dropdown menu with location options.

- handleSubmit(): This function pulls each attribute from the form by pulling the elements id, and then value. A filter object is created as a map storing those attributes. The data is filtered using the filter objects, and generates an html table from that filtered data, which is displayed in a modal.

- eventFilter(events, filterValue, filterFunction): This function takes in an event (our data), a filterValue, and a filter function to filter the events with the provided filtering function.

- colorFilter(event, filterValue): This function takes the event, and filter value, and returns the event if its color attribute matches the filter value.

- bodyFilter(event, filterValue): This function takes the event, and filter value, and returns the event if its body attribute matches the filter value.

- makeFilter(event, filterValue): This function takes the event, and filter value, and returns the event if its make attribute matches the filter value.

- licenseFilter(event, filterValue): This function takes the event, and filter value, and returns the event if its license attribute matches the filter value.

- locationFilter(event, filterValue): This function takes the event, and filter value, and returns the event if its location attribute matches the filter value.

- generateTable(resultObjs): This function takes in the result objs and creates an html template literal to display the data.

- closeButton.eventListener: This event listener sets modalContainer.style.display to none.
- showImageButtons button.eventListener: This event listener will display, or hide the image upon clicking the button.
- generateEntriesIncrement(incBool): This function will generate an HTML table of the data using template literals, and will conditionally generate more entries depending on the incBool variable.
- viewMoreButton.eventListener: This event listener calls generateEntriesIncrement with the inc conditional set to true.
- stripText(text): This function uses a regex pattern to replace white space with a blank space.
- getCarById(id): This function returns the car with the appropriate id.
- generatesFilteredEntries(filteredList): This function creates an HTML table with the filtered list.
- submitButton.eventListener: This event listener creates a filtered data set, and calls generatesFilteredEntries with the filtered data set.
- clearButton.eventListener: This event listener clears out text inputs of the class filter-input.
- deleteEntryById(id): This function deletes the car with the specified ID from the locally stored carList.
- updatecarList(id, changeList): This function updates car from the locally stored carList.
- updateCellData(carId, fieldName, newValue): This function updates the car entry for the remotely hosted data.
- confirmChanges(): This function prompts the user to confirm changes.
- attachCheckBoxEventListeners(): This function attaches event listeners to the check-boxes with the class selectRowCheckbox.
- window.deleteSelectedRows = async function(): This function will delete the selected row from the remotely hosted data.
- async function fetchData(): This function submits a GET request to the JSONBin endpoint to retrive the car data table.
- async function saveData(): This function submits a PUT request to the JSONBin endpoint to save all modifications made to the table as a whole.

- async function addRecord(): This function submits a reuqest to the JSONBin endpoint to add an entry to the car data table, eliminating duplicates using the license plate number. The new records are assigned an ID based off a simple maximum function performed on the current IDs.
- async function updateRecord(): This function updates individual record data when modifications are made via the table in the web application.
- async function deleteRecord(): This function submits a delete request to the JSONBin endpoint to remove an individual record by ID.
- async function run(): This is a function to manually test each of the functions defined in the file.
- async function fetchData(): This function submits a GET request to the JSONBin endpoint. This function is used just to retrive the entirety of the data so that a new record may be appended from the CNN subsystem.
- async function saveData(): This function submits a PUT request to the JSONBin endpoint to update records with modified entries from the CNN subsystem.
- async function addRecord(newRecord): This function submits a request to the JSON-Bin endpoint to add a new record from the CNN subsystem. Duplicates are filtered and removed via license plate number.
- async function processFeedAndAddRecords(): This function parses the JSON output from the CNN subsystem into individual entires which are then added to the JSONBin endpoint.
- scheduleFeedProcessing(): This is a function which is used to set a 5 minute interval between calls of the processFeedAndAddRecords function so that the data from the CNN subsystem is updated every 5 minutes.