S24-20 MDE Team

GDMS – Engine Detection

Final Project Binder

April 26, 2024

Table of Contents

Problem Analysis

Objectives

A US Navy submarine's mission is to remain as silent as possible to complete its objectives. Sailors must continuously measure sound output within the submarine. Combining that complex task with computer assistance would greatly benefit the sailor, the mission, and the entire boat's safety. The goal of this project is to apply machine learning techniques to identify car engines as a proof of concept to be used with submarines in the future.

Requirements

For this project, the customer needs a desktop program that will listen to its environment with a microphone and, when it hears a car engine, make a prediction on the most likely car brand to have produced that noise. The customer has specified that the sensing part of the application must only activate upon hearing engine noise. Furthermore, the AI algorithm must be able to listen to the engine noise and make a prediction within a total of 5 minutes and use narrow-band analysis to gather and identify data. The application must also show the correct car brand as its top prediction with at least 80% accuracy. Finally, the application in its entirety must work without an internet connection.

The customer has also identified various stretch goals for this project. First, the project should be able to show the correct car brand as its top prediction with at least 90% accuracy. Second, the application should display the 3 most likely car brands to have produced the engine noise in a GUI rather than the terminal. Lastly, the project team should conduct a study on how the distance between the engine and the microphone

affects the accuracy of the AI algorithm. The degrees of accuracy should be displayed based on the distance measurement for a given audio recording.

Req. #	Description
CR-1	The system must activate only after hearing engine noise
CR-2	The system must listen and predict the most likely car brand within 5 minutes
CR-3	The system must use narrow-band analysis to gather/identify data
CR-4	The system must make predictions with at least 80% accuracy
CR-5	The system must work completely without an internet connection
CR-S1	The system should make predictions with at least 90% accuracy
CR-S2	The system should display the top 3 predicted car brands inside a GUI
CR-S3	The system should display degrees of accuracy for predictions given a distance measurement between the engine and the microphone

Table 01-1: Customer Needs

Constraints

External Factors - Global, Cultural, and Environmental

Since car models and car makes vary around the world, different countries can have specific rules and regulations they need to uphold. For instance, road noise policies vary from regions like Australia, Europe, and the US. According to a paper in the National Library of Medicine, “noise limits for emission scopes (exhaust and engine noise combined) were more concentrated in Europe (max – min = 15 dB(A)) than those identified in Australia (max – min = 29 dB(A)) and North America (max – min = 31 dB(A))” [1]. Since we will be gathering noise samples from many different car brands, we may need to consider global noise regulations in our AI/ML algorithms.

As stated above, noise preference and regulation vary from country to country. While this is a global matter, it can also be considered a cultural one as well. Since every culture has its own beliefs and ideas, this affects regulations like road noise. Seeing how these factors relate to each other, we will need to be more aware of and acknowledge how they can impact our project.

Our model will be designed to recognize the engine noise of cars. From an environmental perspective, this project has the potential to mitigate unwanted noise, which ultimately could address larger issues like noise pollution. In doing so, it could help

with the negative effects of noise pollution like “stress-related illnesses, high blood pressure, speech interference, hearing loss, sleep disruption, and lost productivity” [2]. However, among these, the most common health issue is Noise Induced Hearing Loss. According to the EPA, this is caused by hearing constant or loud noise for a prolonged amount of time. By reducing these impacts, our project could be used to improve the overall quality of life for people in noise-affected areas.

Social Factors - Public Health, Safety, and Welfare

On a smaller scale, the engine noise detection program has a very small niche area of focus. The program ideal is constantly listening for an engine idling in its environment and then outputs the possible brand or model of the car. Possible safety concerns such as using the program to identify expensive cars that could be used for stealing. There are not many health concerns as the main portion of the project is done on software. The program is an audio detection program, which can lead to issues with privacy invasion if the program is constantly listening to its surroundings and can lead to safety and welfare issues.

With the main portion being on the computer, it is not necessary to use a physical lab. Although there can be programs that run for example test cases and possible experiments with a microphone outside. Following the recording, there should be a proper system to record cars or retrieve a dataset of idling engine noises. The documentation specifications include that for recording the car, the car must be stationary and parked.

Audio detection can be very useful when used correctly as it can provide insight and information on people and objects. For example, are security cameras recording the surroundings or using audio detection to pick on peoples’ voices through their devices? While the project is meant for cars, the bigger picture of being able to use AI and ML to detect types of specific sounds can lead to safety concerns and public health issues.

Ethical Factors - Global, Societal, Economic, and Environmental

From IEEE Code of Ethics [3], Part Ⅰ.1 “*strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment*” This section is relevant to our project as we are creating a system that requires sound samples from individuals personal vehicles which should be a Privacy consideration. Additionally, we are developing a system of detecting specific engine types and if misuse in a certain way could be a danger to the public. This possible misuse could also take place in another relative part of the ethics code Ⅱ.7 which states “*to treat all persons fairly and with respect, and to not engage in discrimination* “[3].

One of the reasons these ethics could have an impact on the development of our project is the concern about people's privacy. Designing a program to identify a car based on its engine sound seems harmless enough however it could quickly become a problem if the program gets too specific in its results. For example, if it not only told the user the year of a car but also information about the maintenance level or other highly specific data of the car then a person could use our program to pinpoint an exact user and determine whenever they are using their car.

Additionally, we want to make sure that our project is not used to target or discriminate against any group of people. For example, if a law enforcement officer were to use our program to identify whenever a high-class sports car is in the area so they can keep a close eye on them and make sure they aren't speeding this would cause a problem. And while we have no control over how our customers may choose to use our program we should still do our best to mitigate the amount of inequality that can be caused by our design. The final reason these ethics greatly impact our program's design is the idea of criminals. In a similar vein to having little control over how our project is used, comes the idea that we need to consider what could happen if a criminal chooses to use our program to achieve some malicious goal. For example, if a thief attempts to use our program to locate old or abandoned cars that are easier for them to steal. We need to consider how to deter this kind of behavior and design this project in such a way that we minimize the possible negative effects it could have on the general public regardless of the user.

Overall before our group dives too deep into this project we need to stop and consider the moral responsibilities that come along with it. We need to take a step back and think of every possible way this simple AI program could be used to perform malicious and unethical acts and then take every step possible to mitigate these possibilities. As well as making sure that our program can target any one specific person/group and do the best we can to ensure that even if someone attempts to use our project for hurtful activities the damage done is minimal.

Engineering Standards

Req. #	Requirement	Source Document (e.g., standard, regulatory requirements)	Details

SR-1	Engine Sound Level Measurement Procedure	SAE J1074_201405	Sections 5 through 8 outline a consistent method of obtaining engine sound readings from a car
SR-2	IEEE Standard for Advanced Audio Coding	IEEE 1857.2-2013	Section 3 provides an in-depth overview of both lossy and lossless audio encoding
SR-3	BS ISO/IEC 13818-1. Information technology. Generic coding of moving pictures and associated audio information. Part 1. Systems	MP3 (MPEG-2 Version 3) - ISO/IEC 13818-1	The document provides information on audio encoding

Detailed Design

System Architecture Diagram

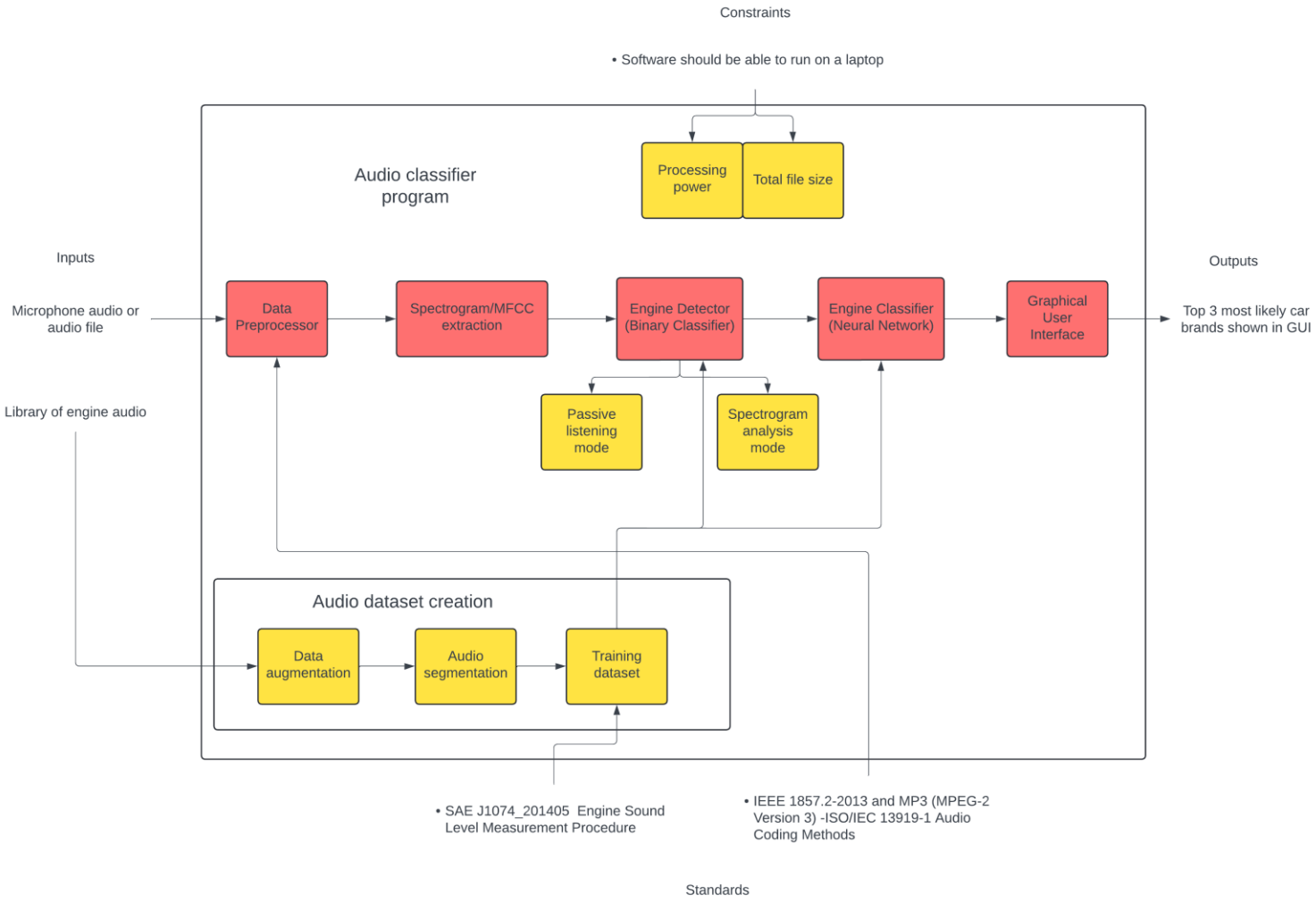


Figure 1: The system architecture diagram for our solution. The red boxes represent the various subsystems outlined in the Detailed Design section.

Data Preprocessor Subsystem

This subsystem standardizes data received, ensuring that all audio data maintains consistent dimensions, including length, sample rate, and stereo audio. This standardization is necessary due to the wide variety of microphones that may be used. Without this uniformity, mixing audio dimensions could negatively impact the accuracy of the neural network.

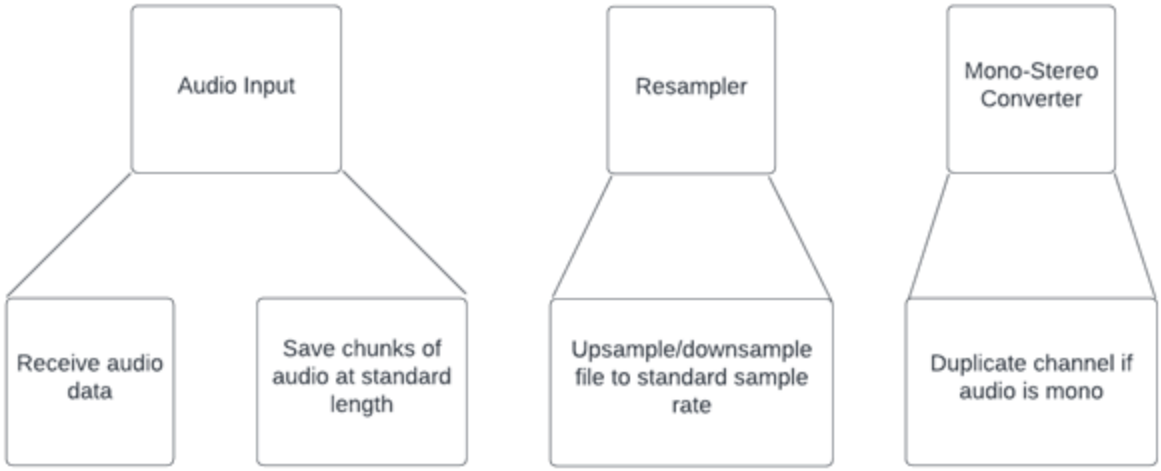


Figure 2: Data Preprocessor Subsystem diagram

The preprocessor is designed with specific functionalities to optimize performance. It saves audio files in 5-second chunks, ensuring low memory usage of approximately 15MB at a rate of around 3MB/s. This approach provides sufficient data for the neural network's needs. Additionally, the preprocessor resamples audio files at 48Hz if necessary, as this is the standard audio sample rate for Zoom H1 microphones. It also converts audio to a stereo channel if needed, ensuring consistency with the neural network training data.

The algorithm for processing audio involves several steps. First, it records 5 seconds of audio from the user-selected microphone. If the recorded audio doesn’t have a sample rate of 48kHz, it resamples the audio using librosa.resample(). In cases where the audio is mono-channel, it transforms it into stereo using AudioSegment.from_mono_audiosegments. Finally, the algorithm saves the processed audio to an output file as an uncompressed .wav file.

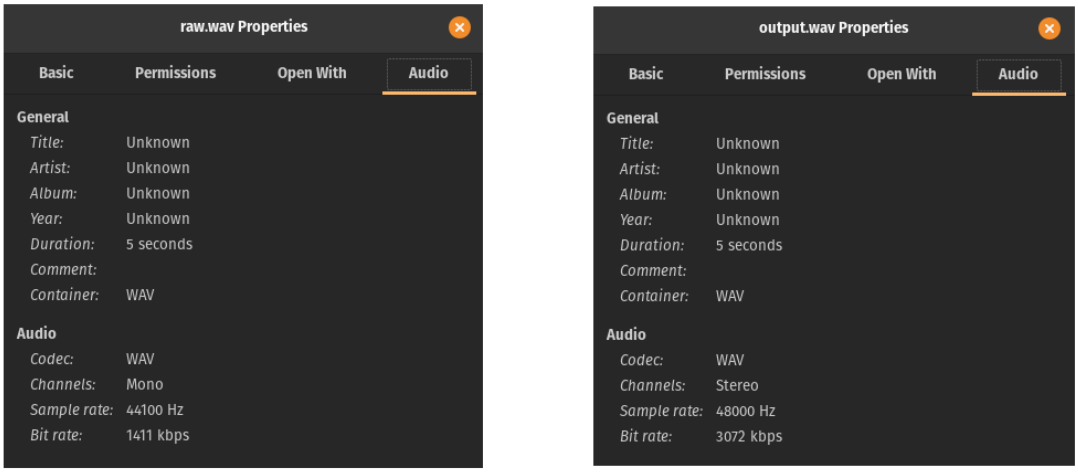


Figure 3: Proof of concept testing our data Preprocessor with results before and after going through the Preprocessor

Spectrogram/MFCC Generation Subsystem

The system incorporates the Python library Librosa, a third-party tool, to generate a mapping of signal strength over specific frequency ranges. This output is then used as the input for the machine learning model's predictions

```python
def extract_mfccs():
    data = {
        "train": {"mfcc": [], "labels": [], "category": []},
        "test": {"mfcc": [], "labels": [], "category": []},
    }

    sample_rate = 48000
    n_mfcc = 13
    n_fft = 2048
    hop_length = 512
    expected_num_mfcc_vectors = math.ceil(sample_rate / hop_length)

    # Loop over each brand in the cut audio train folder
    print('Extracting training MFCCs...')
    for i, brand in enumerate(chunk_train_output_dirs):
        brand_files = os.listdir(chunk_train_output_dirs[brand])

        # Loop over each audio file in the brand folder
        for file in brand_files:
            # Get the file name without the extension
            #file_no_ext = re.split(r'.' + AUDIO_FILE_EXT, file)[0]

            # Load the audio file
            file_path = r'{}/{}'.format(chunk_train_output_dirs[brand], file)
            signal, sr = librosa.load(file_path, sr=sample_rate)

            # Extract the MFCCs from the file
            mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_fft=n_fft, n_mfcc=n_mfcc, hop_length=hop_length)
            mfcc = mfcc.T.tolist()
```

Figure 4: Code to extract MFCC from the audio recordings to be able to be used in the Engine Detection and Engine Classification Subsystems

This subsystem outputs into a sort of generated spectrogram known as Mel Frequency Cepstral Coefficients (MFCCs) which is used in machine learning. It creates a spectrogram but instead of visually representing the audio it uses numbers which is easier for machine learning models to interrupt. This creates a list of MFCC vectors for each input audio file. An example of these vectors is shown below. Numbers closer to the top represent the energy of low frequencies within the audio file while numbers closer to the bottom represent higher frequencies.

```
-278.6817932128906,
144.55056762695312,
-50.711761474609375,
-12.547877311706543,
-20.881235122680664,
12.194741249084473,
5.302641868591309,
15.229799270629883,
-3.250492572784424,
8.266886711120605,
-2.7368788719177246,
-1.3312960863113403,
-7.529457092285156
```

Figure 5: MFCCs generated using previously shown Python code

The accuracy and consistency of the MFCC generator shall be tested using the neural network due to how closely they are coupled. That is, the neural network should have accurate and repeatable results if the MFCC vectors it receives as input are correct.

Machine Learning Detection Subsystem

The Machine Learning Detection subsystem is an important part of our system. This subsystem handles whether the audio that is being heard is an engine idling or not. It does this by constantly listening to its surroundings in 5-second audio clips. It then takes these 5s and splits them into 1s audio clips. It then extracts the MFCCs from each clip and runs them through a binary classification model. If 2 or more of the 1s clips are detected as an engine idling, we then declare an engine has been detected and then allow the classification system to classify the audio.

To train our model, we created a database of car engines idling vs background ambiance noise. To create this, we recorded audio ourselves, as well as scraped it off of the internet. Once we had this audio, we segmented it all into 1-second audio clips. In the end, we made sure to have a balanced dataset by trimming the sets to have the same number of training data.

Database creation

Engine Idling

Our Own Recordings Parameters

- ∉ Engines idling
- ∉ The car engine hood open

- 5 min recordings
- H1 Zoom mic on a tripod as close to the front of the car with the mic slightly above the surface parallel to the engine.
- wav format

Audio Scraped from Online

- Engines idling
- The car's year make and model if we know them.
- Trimming the audio to just be the car idling

Background/Ambiance Noise

Our Recordings

- H1 Zoom mic
- Anything
  - Walking
  - Talking
  - White noise recording of an outdoor environment
  - Rain

ESC-50

- Extracted environment sounds from Github
- Includes 50 different environment noises with 5-second clips in 2000 samples

Online audio

- Audio extracted from YouTube of background ambient noise

Label	1 Second Clips
Label 0 (Background)	148,050
Label 1 (Engine Idling)	148,050

Table 2: Engine Detection Database

Each of these 1-second audio clips is then labeled and turned into MFCCs for model training. For MFCC extracting we have picked 13 MFCCs and because our wav files are recorded at 44,000hz, for each 1-second audio clip we get 94 MFCC segments with 13 MFCC per segment. Thus our data is structured as 1 second audio clips, 94 MFCC samples, and 13 MFCCs, so we get an array the size of 296100,94,13. However, binary

classification models take 2d arrays, so for each 1-second audio clip we average the 94 MFCC samples. Thus our final dataset used for training is a 1D array for labels of size 296100 and a 2d array for features with the size of 296100,13.

Dataset Visualization

The dataset's variance between features is very high due to the strong difference between audio recordings. The variance of Label 1 is far less than Label 0, as engines idling are more closely related, while the background noises cover more of a variance of different sounds and noises.

Label	Variance
Label 0 (Engine Idling)	13,375.484
Label 1 (Background)	23,940.162
Dataset Total	18,667.057

Table 3: Data Collection Database for Engine Detection

When plotting the dataset in a 2D space using Principal Component Analysis (PCA) and scaling the data using Z-score (where the mean of the data is subtracted from each data point and then normalized to have a mean of 0 and a standard deviation of 1), it becomes challenging to distinguish between Label 1 and Label 0 as there is no clear boundary. PCA is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space. It achieves this by identifying the directions, or principal components, that capture the maximum variance in the data. By retaining only the most informative components, PCA simplifies the dataset while preserving as much variance as possible, making it easier to analyze and visualize complex data. In our case, we are turning the mean average of 13 MFCCs per 1s audio clip down to a 2D space rather than a 13D dimensional space which we can't simply view. It is hard to see here but the orange (Label 0) and blue (Label 1) dots are layered on top of each other so there are blue dots all below the orange dots. It almost appears as if engine idling is a subset of the outer background noise as the orange dots appear within the larger spread of the blue dots. This graph also explains our variance more as you can see orange dots are more closely related.

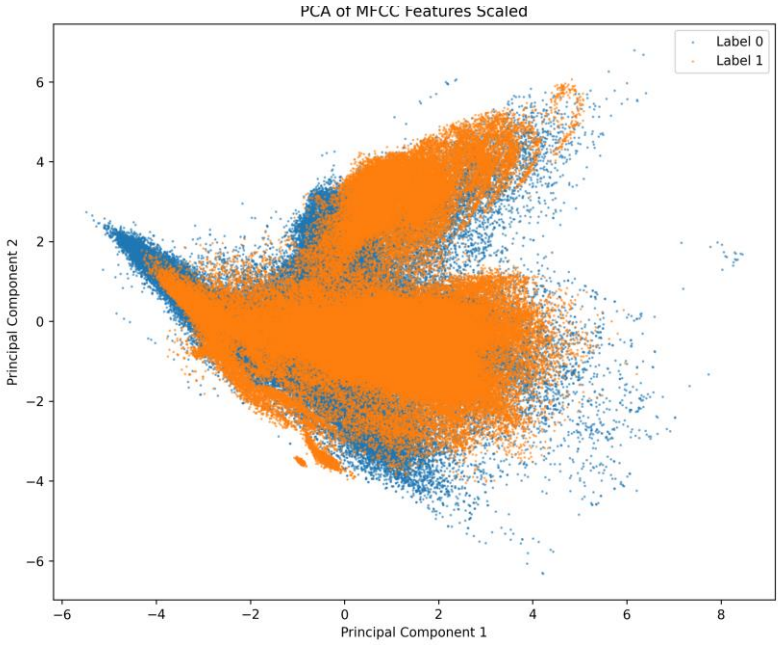


Figure 6: Principal Component Analysis of Engine Detection Database

Binary Model Description

The binary classification model for our engine detection is known as a Random Forest Classifier. Random Forest is an ensemble learning method that constructs a multitude of decision trees during training. Each tree in the forest independently predicts the output, and the result is determined by a majority vote. This method is effective for tasks like binary classification, where it combines multiple decision trees to enhance predictive accuracy.

The model was trained on our “engine vs not engine” dataset and we used an optimization algorithm to find the best model hyperparameters such as n_estimators, max_features, and min_samples_leaf. The best parameters found were:

```
RandomForestClassifier(n_estimators=500, maxs_features = 'sqrt', min_samples_leaf = 1, random_state=42, n_jobs=-1)
```

Furthermore, we used a Kfolds cross-validation approach with 10 folds and we saved the highest performing model out of the 10.

In this PCA representation of the dataset, you can see the decision boundary created by the Random Forest classifier.

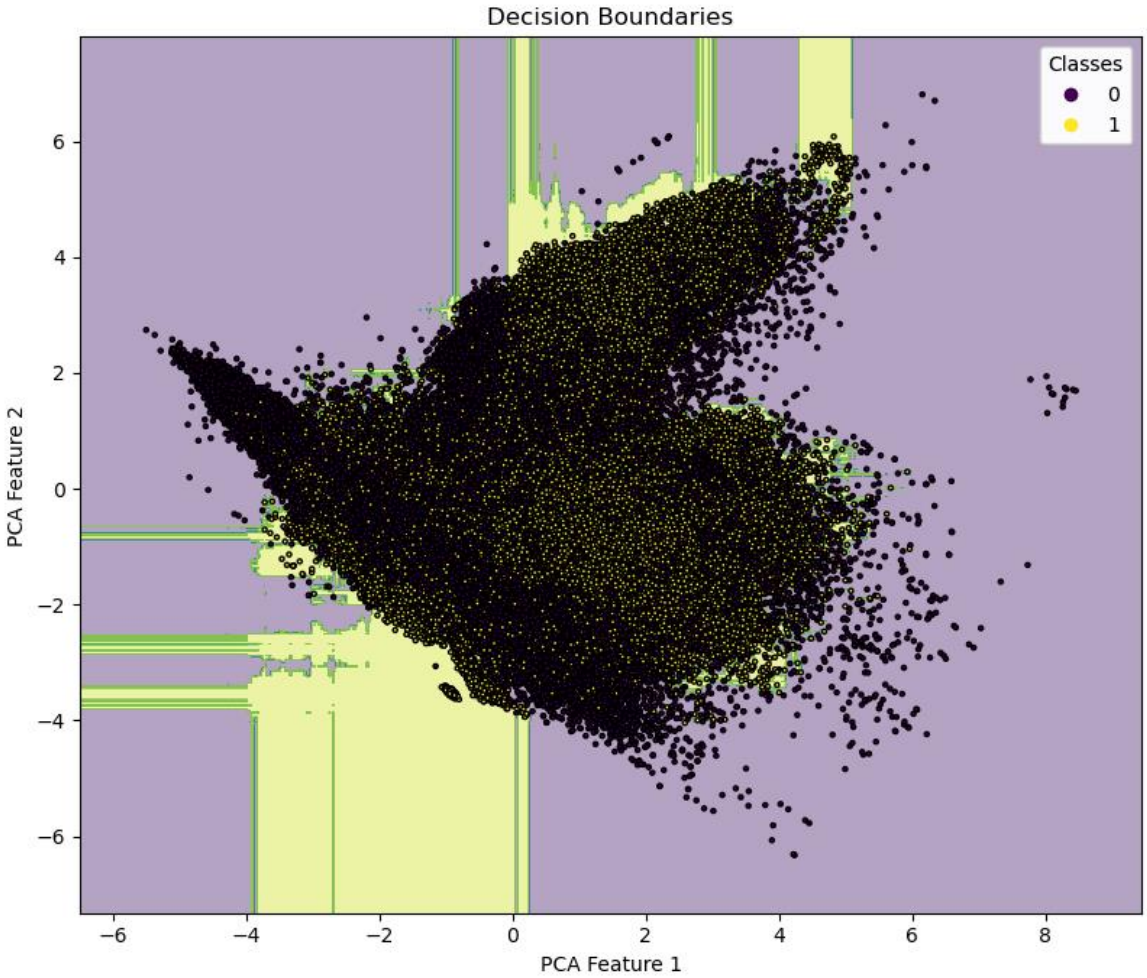


Figure 7: Representation of the Random Forest Decision Boundary on a PCA Representation of the Dataset

Our Random Forest classifier scored a 98.20% accuracy on average when run on the Kfolds test. Below is the classification report of the model and its corresponding confusion matrix, allowing us to identify instances where the model made false positives and false negatives. False positives occur when the model incorrectly predicts a positive outcome that is actually negative, while false negatives occur when the model incorrectly predicts a negative outcome that is actually positive.

	Precision	Recall	F1-Score	Support
Label 0	0.98	0.98	0.98	37019
Label 1	0.98	0.98	0.98	37006
Accuracy			0.98	74025
Macro Avg	0.98	0.98	0.98	74025
Weighted Avg	0.98	0.98	0.98	74025

Table 4: Classification Report from Random Forest Classifier

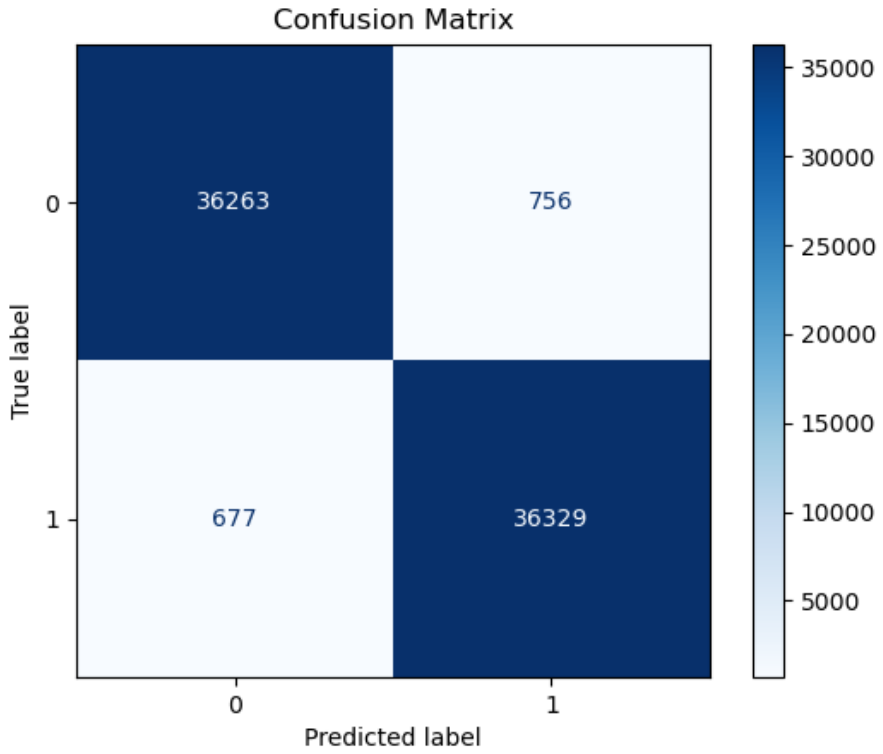


Figure 8: Confusion Matrix from Random Forest Classifier

Binary Model Testing

To test our model, we recorded a validation set completely separate from any data used to train or test our model. Thus, the model would have never seen this data until testing. This validation set consisted of 10 minutes of engine idling with the microphone at different distances away from the engine and then 10 minutes of ambient outdoor noise. The random Tree model achieved an 81.7% accuracy on this validation set. Showing that the model wasn't overfitted and was able to work on new data. The results from the test are shown in the table below:

label	Distance	Accuracy	F1-Score	Recall	Precision
Label 0	5 feet	0.729	0.843	0.729	1.0
	1 foot	0.742	0.852	0.742	1.0
	Bumper	0.970	0.985	0.970	1.0
	On engine	0.869	0.826	0.942	0.735
Label 1	Front of car	0.833	0.0	1.0	0.0
	Back of car	0.676	0.0	1.0	0.0
	Music	0.893	0.0	1.0	0.0
Overall		0.817	0.768	0.849	0.700

Table 5: Results for Running Random Forest on Validation Set

Choice of Binary Model

Why did we pick Random Forest as our model? We tested a total of 16 different classification models and 6 Neural Networks(NN).

Classification Models Tested	Neural Networks Tested
GaussianNB	Complex Convolution Neural Network(CNN)
MultinomialNB	Simple CNN
RandomForest	Complex Recurrent Neural Network(RNN)
LogisticRegression	Simple RNN
SVM	Complex Deep Neural Network(DNN)
GradientBoosting	Simple DNN
Ridge	
PassiveAggressive	
Perceptron	
SGD	
KNeighbors	
DecisionTree	
ExtraTrees	
Bagging	
AdaBoost	
QuadraticDiscriminantAnalysis	

Table 6: Listed the Machine Learning Models Used

During the Kfolds training and test, we observed that Extra Trees was the best with an accuracy of 98.57%. Then in the validation set, Quadratic Discriminant Analysis classification model had the highest accuracy, 86%. However, after further testing I Could Tell Quad had only learned that any loud noise was an engine idling. Extra Trees also had a similar accuracy to that of Random Forest in the validation set. However, Extra Trees did very well at the normal recording distance that we had done most of our recordings from but struggled at other distances. Thus, I decided to go with Random Forest as it was less overfitted to this exact distance than Extra Trees was. Going with Random Forest also gave the bonus of having a smaller file size of 148Mb vs Extra Tree's 469Mb. Below are the decision boundaries for Extra Trees and Quad. You can see from this, compared to Random Forest that there is less decision boundary space, for if new data were to come in, it wouldn't know for Extra Trees. Then for Quad, you can see how simple the decision boundary is, explaining how the model must have found a difference between engine and not engine, which was just the change in loudness, not determining if the sound it was hearing was an engine or not.

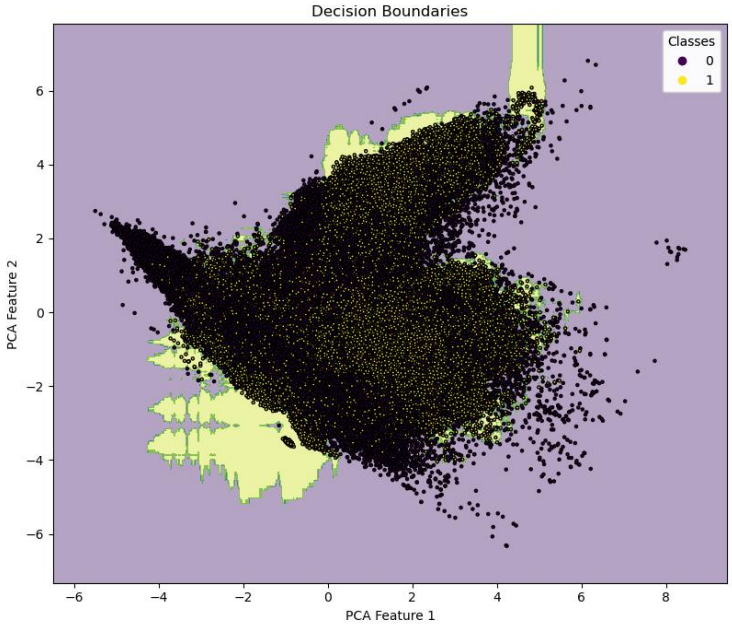


Figure 9: Representation of the Extra Trees Decision Boundary on a PCA Representation of the Dataset

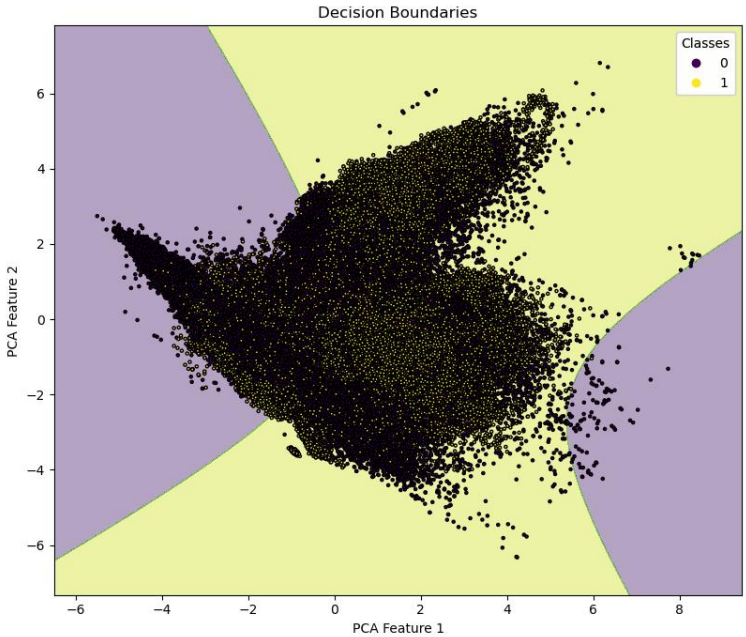


Figure 10: Representation of the Quadratic Discriminant Analysis Decision Boundary on a PCA Representation of the Dataset

Machine Learning Classification Subsystem

The Machine Learning classification subsystem is an essential component of the system, as it directly handles the identification of car brands. This subsystem functions by taking in the generated MFCCs as input. It then runs these MFCCs through a pre-trained model, which is designed to return the three most likely car brands based on the audio analysis.

The model is trained on an audio database we have created consisting of recordings of various car engines idling. We curated this data set from our own recordings we made with the H1 Zoom mic as well as scraping cars idling off YouTube. As you can see from the table below, we were only able to record 50 cars ourselves and found 176 online. Although our recordings were of higher quality, not compressed, and 5 minutes in length, the YouTube scrapped recordings were shorter and of less quality.

Brand	Our Recordings	Online Recordings	Total	Total Length of Recordings in Seconds
BMW	1	18	19	810.39s
Ford	2	12	14	1,123.76s
GMC	3	19	22	1,254.59s
Honda	10	17	27	3,514.63s
Hyundai	3	15	18	1153.50s
Jeep	0	18	18	527.52s
Kia	1	17	18	1,097.82s
Nissan	5	17	22	1,954.19s
Subaru	0	22	22	1,121.10s
Toyota	25	21	46	4,141.92s
All Brands	50	176	226	16,699.42s

Table 7: Data Collection Database for Brand Classification

Dataset Visualization

Label	Variance
BMW	9763.2841796875
Ford	7813.3203125
GMC	10651.9921875
Honda	15278.1826171875
Hyundai	10172.234375
Jeep	8640.78515625

Kia	9252.46484375
Nissan	9774.8662109375
Subaru	11430.0673828125
Toyota	11713.5947265625
Dataset Total	11370.407

Table 8: Data Collection Database for Brand Classification

The graph below is of a t-distributed Stochastic Neighbor Embedding (T-SNE), this is a dimensionality reduction technique that, unlike PCA, performs non-linear dimensionality reduction. It's commonly used for visualizing high-dimensional data in lower dimensions. In the graph below, each interconnected cluster of dots represents a distinct recording of a car, with different colors distinguishing between brands. Notice how there isn't a clear separation indicating all instances of a particular brand. Instead, the data points are spread out, suggesting that cars of the same brand exhibit diverse characteristics. Additionally, you'll observe that there's no apparent correlation between even similar models, such as a 2010 Honda Accord and a 2012 Honda Accord are nowhere near each other (You will have to take my word for it. It is hard to display the graph on paper, it is better in 3d and when interactive).

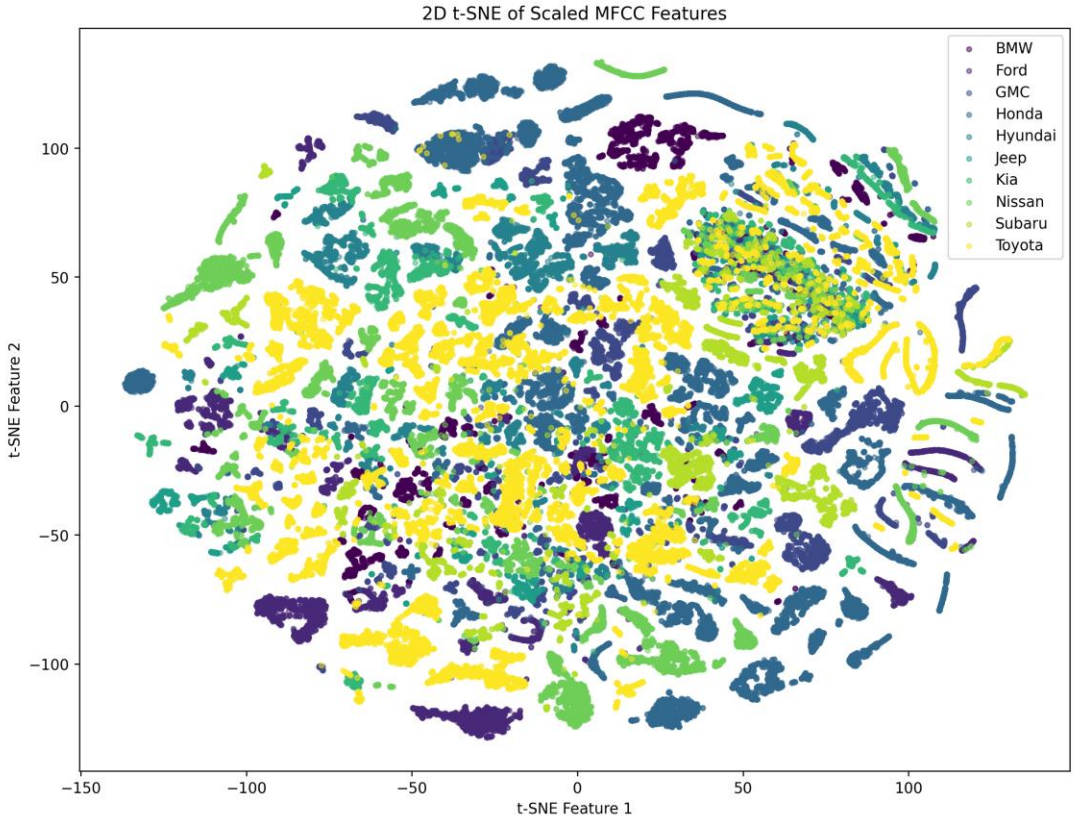


Figure 11: T-SNE of Engine Classification Database

Neural Network Description

The neural network we will use shall be a Fully Connected network constructed using the Keras Python library, consisting of a flattening input layer, three fully connected hidden layers, and a fully connected output layer. All hidden layers will use a ReLU activation function and have a dropout of 0.3 to help prevent overfitting. The output layer will use a softmax activation function since there are multiple car brands to choose from.

To compile the network, we will use the Adam stochastic gradient descent algorithm as the optimizer with Sparse Categorical Cross Entropy as the loss function. We will train our neural network on the input data using a batch size of 32 and stop training when the model's validation loss doesn't improve for 5 epochs.

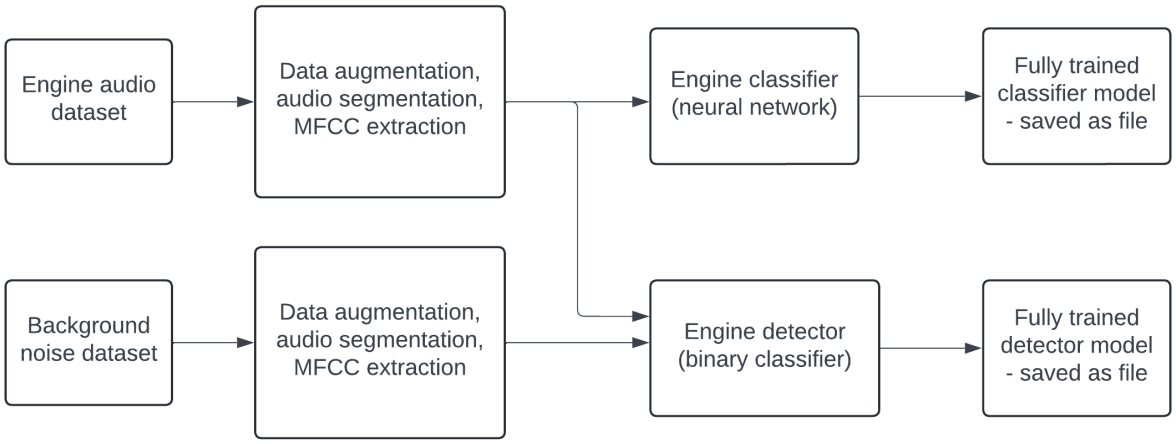


Figure 12: Machine Learning Model Subsystem training architecture diagram

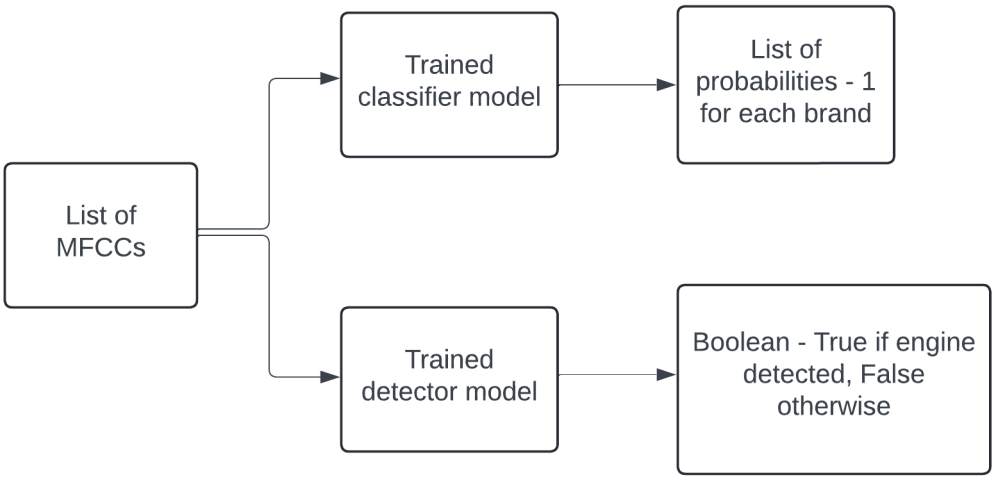


Figure 13: Machine Learning Model Subsystem post-trained architecture diagram

```
# Specify the neural network's architecture
model = tf.keras.Sequential([
    # Input layer
    tf.keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2])),
    tf.keras.layers.Flatten(),

    # First hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),

    # Second hidden layer
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.3),

    # Third hidden layer
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.3),

    # Output layer
    tf.keras.layers.Dense(len(unique_targets), activation="softmax")

])
```

Figure 14: Our engine classifier model's architecture

```
# Compile the network
optimizer = tf.keras.optimizers.Adam(learning_rate=0.000_223_869)
model.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
model.summary()

# Train the network
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor the validation loss
    patience=5,  # Number of epochs with no improvement after which training will be stopped
    verbose=1,  # Output a message for each early stopping
    restore_best_weights=True  # Restore model weights from the epoch with the best value of the monitored quantity
)
history = model.fit(inputs_train, target_train,
          validation_data=(inputs_test, target_test),
          epochs=100,
          batch_size=32,
          callbacks=[early_stopping])
```

Figure 15: Our engine classifier model's training code

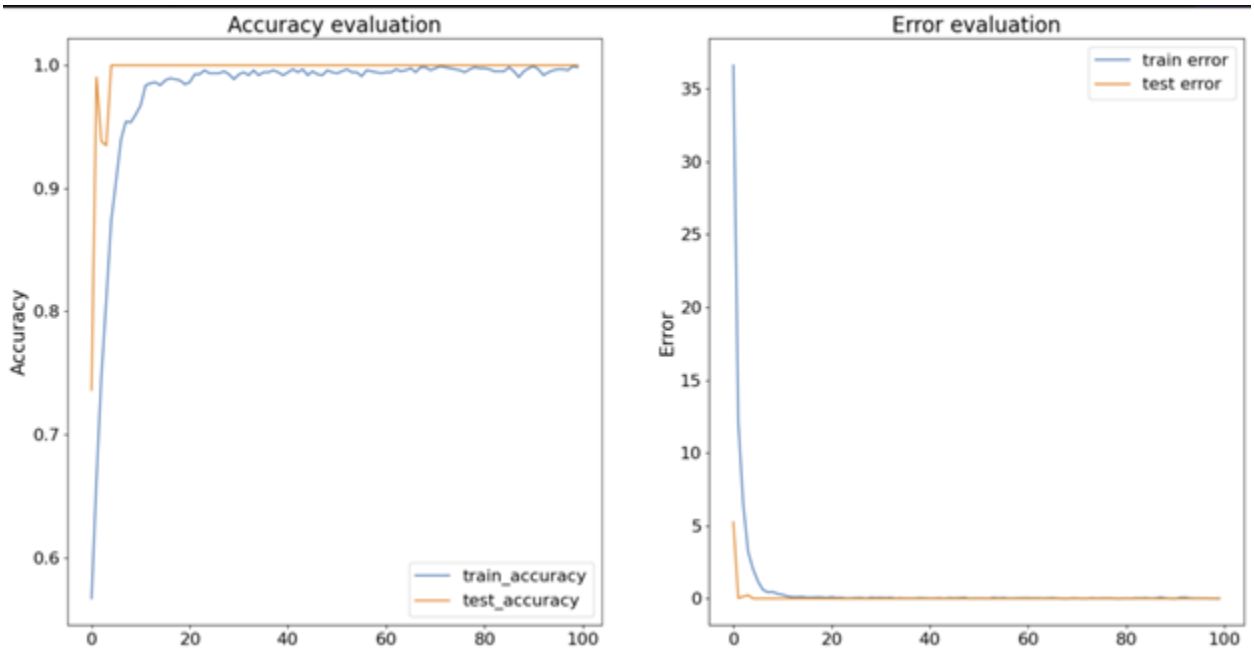


Figure 16: These graphs show our proof of concept testing on just our 5 cars with accuracy and error graphs

As a proof of concept, we chose to test the neural network by having it classify the audio of each team member's cars. The audio files used for testing were all approximately 15 seconds long and were not included in the neural network's training set. To test the

neural network, we created a program that first calculates the Mel-Frequency Cepstral Coefficients of the audio file, then inputs the coefficients to the neural network. The output is a list of probabilities for each car brand, and our model's prediction is taken as the brand with the greatest probability. For this simple task, our model had an accuracy of 100%.

Cars used:

1. Scion TC
2. Toyota Camry
3. Nissan Altima
4. Lexus GS350
5. Toyota Highlander

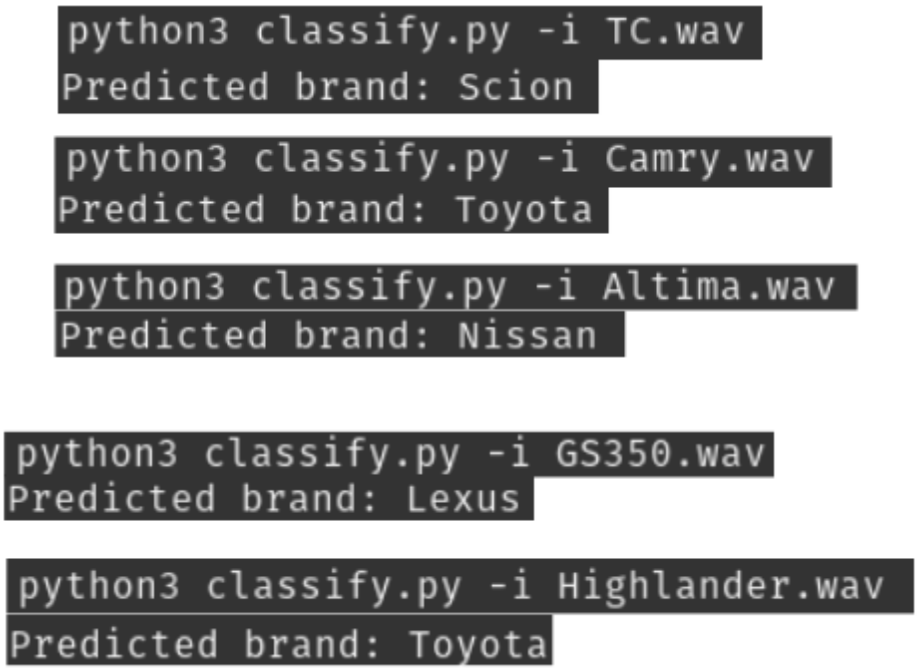
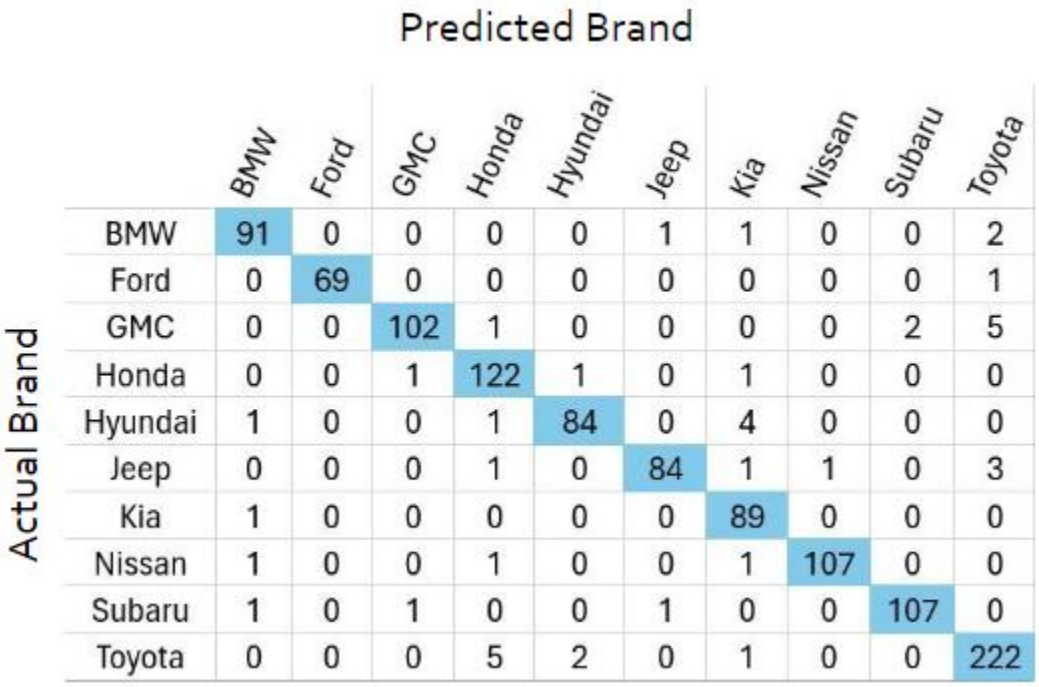


Figure 17: Neural network predictions for various engine audio recordings

After creating the final dataset, we trained the engine classifier using an 80/20 train/test split. This means that, at the beginning of training, 20% of the audio samples were split from the full dataset to be used as a validation set. This approach allowed us to see the model's accuracy as it was being trained, which is very useful due to how long it takes to train the classifier. Using this approach, we found that the final model was able to reach an accuracy of 96%, which meets the required 80% accuracy.

In addition, we tested the model on a new audio set created by taking the initial dataset and applying data augmentation with random parameters. Doing this resulted in an accuracy of approximately 95%. The confusion matrix generated from this test is shown below.



Actual Brand \ Predicted Brand	BMW	Ford	GMC	Honda	Hyundai	Jeep	Kia	Nissan	Subaru	Toyota
BMW	91	0	0	0	0	1	1	0	0	2
Ford	0	69	0	0	0	0	0	0	0	1
GMC	0	0	102	1	0	0	0	0	2	5
Honda	0	0	1	122	1	0	1	0	0	0
Hyundai	1	0	0	1	84	0	4	0	0	0
Jeep	0	0	0	1	0	84	1	1	0	3
Kia	1	0	0	0	0	0	89	0	0	0
Nissan	1	0	0	1	0	0	1	107	0	0
Subaru	1	0	1	0	0	1	0	0	107	0
Toyota	0	0	0	5	2	0	1	0	0	222

Figure 18: Confusion matrix from engine classifier test

The next test was to verify that the engine classifier could produce results within the requirement of 5 minutes. We tested this by feeding each audio file in our training set into the model and recording the average time it took to produce a result. From this test, we found that the classifier took an average of 8.5 seconds to predict the audio's car brand. This is within 5 minutes, so it meets the requirement.

Graphical User Interface Subsystem

The Graphical User Interface Subsystem is needed to display the output from the machine learning subsystems to the user. It allows the user to select an audio input, either a microphone or an audio file and displays the top three predicted car brands in order.

Below is an initial sketch of our GUI. After receiving feedback from the customer, we removed the spectrogram window as it wasn't necessary for the project. In addition, we added the ability to read audio from a file since it made the program more flexible.

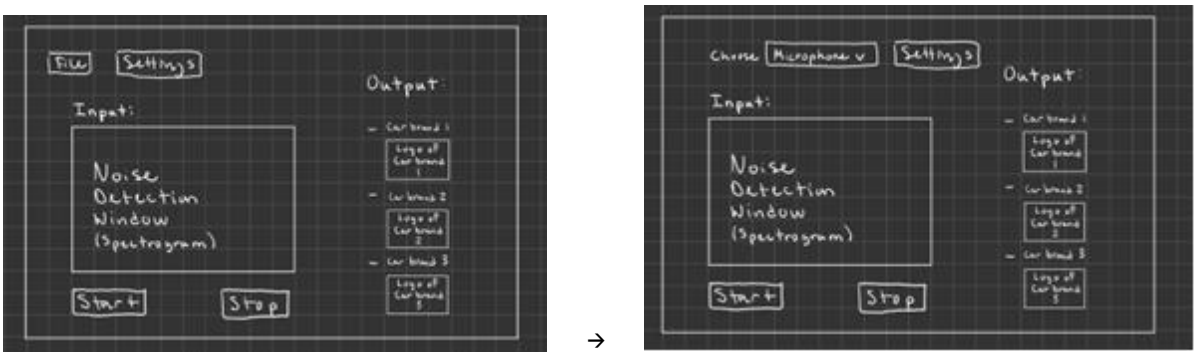


Figure 19: Sketched storyboards for the graphical user interface

Figure 20 shows the final GUI. It allows the user to select a microphone or file to take audio from, with a status bar to report any errors encountered in reading the audio. Once the user has selected an input, they must press the "Start Analysis" button. After that, the program will record 5 seconds of audio if a microphone was selected, or read a 5-second segment of audio if a file was selected. This audio is then fed into the engine detector. If it detects an engine in the audio, then the data will be sent to the engine classifier which will show the top 3 predicted car brands. The program then starts recording more audio. At any point in this process, the user can press the "Stop Analysis" button to stop the program.

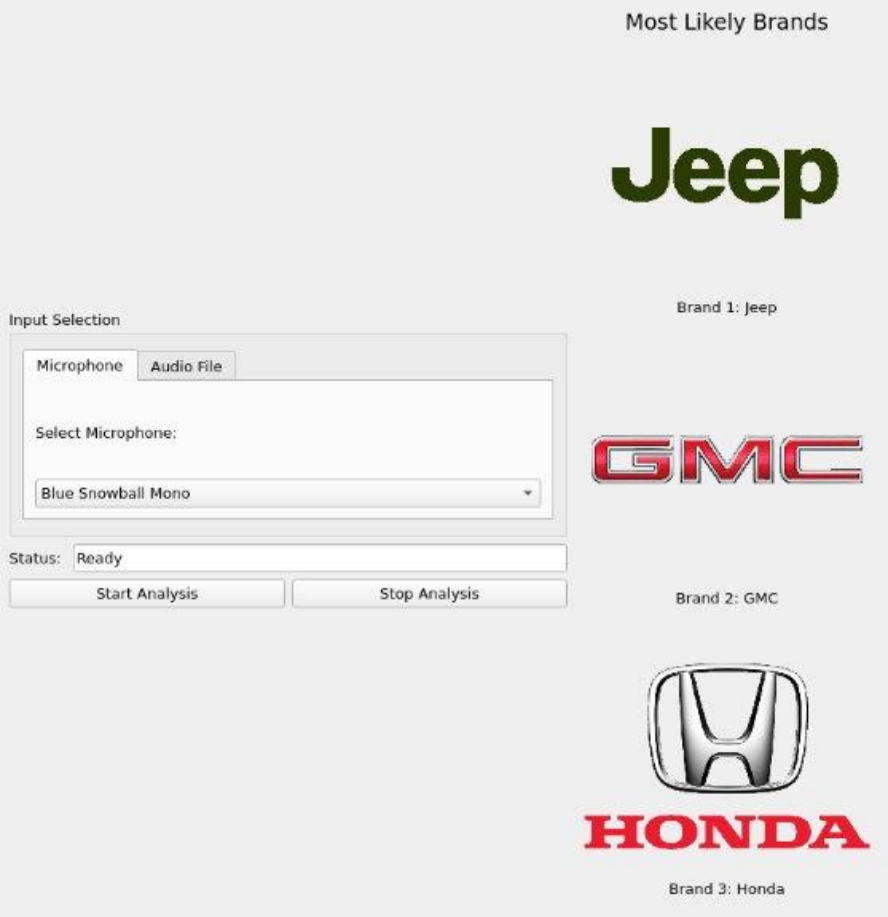


Figure 20: Graphical User Interface implemented using PyQt

Build and Test

Project Status

At the time of writing, our project is finished. We have completed the engine classification program and presented our project at the MDE Expo.

Test Results/Analysis

Through our tests outlined in the Detailed Design section, we were able to determine the following:

- Engine classifier is only activated when an engine is detected
- Engine classifier has 96% accuracy (required 80%)

- The program makes predictions in approximately 8.5s on average (required < 5 minutes)

For further testing, we tested our model in the real world on 5 cars it was trained on and 2 new cars that it had never seen before in the dataset. Of the 5 cars, our model successfully identified the correct brand among the top three predictions for all 5 cars it was trained on. Whether it was the first or second prediction, the correct brand consistently appeared within the top ranks of confidence. However, when presented with two new cars not included in the training set, yet closely resembling models present in it (such as a 2010 Honda Accord in the training set and a 2016 Honda Accord), the model was unable to determine the brand. This, coupled with the absence of clear distinctions between brands when visualizing the data as depicted in the T-SNE representation, suggests that our model may have become overfitted. To test this theory, we trained the model again, but this time instead of doing an 80/20 split, we held 5 models out of the training set for each brand. This is because the 80/20 data split is splitting up data that has already been augmented 6 times and segmented into 0.1s audio clips. Thus, for example, during an 80/20 split, the model could be taking a recording of a Toyota Tacoma with 100 0.1s audio samples, taking 80 of them for training and 20 for testing. Thus, that way when the model performs its validation test, it goes “Hey I have seen this Tacoma before, it must be that Tacoma”. This is an example of testing and training data leakage. It was practically training and testing on the same data. As once again shown by the T-SNE graph, MFCC data points are grouped by the exact same recording year, make, and model instead of any visible grouping by make. Thus, by taking out 5 models from each make, our hope was that the model would become more generalized and learn what makes a brand, a brand, rather than just memorizing the year, makes, and models it was trained on. In doing this, the model training failed at 6 epochs. It was bringing down the training loss and increasing the training accuracy. However, the validation loss continued to increase, and no minimum

was found. As a result, the highest validation accuracy achieved was only 20%.

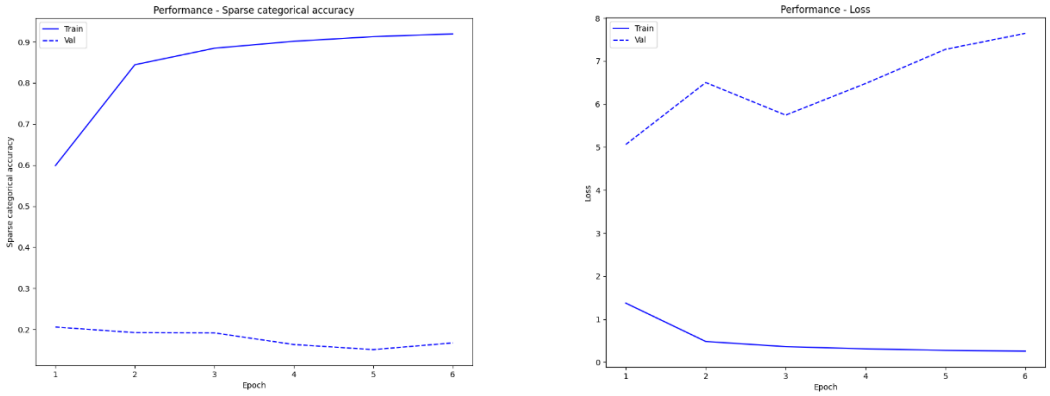


Figure 21: Graph of Training Accuracy and Training Loss vs Validation Accuracy and Validation Loss for the NN using 5 models held out per brand for validation.

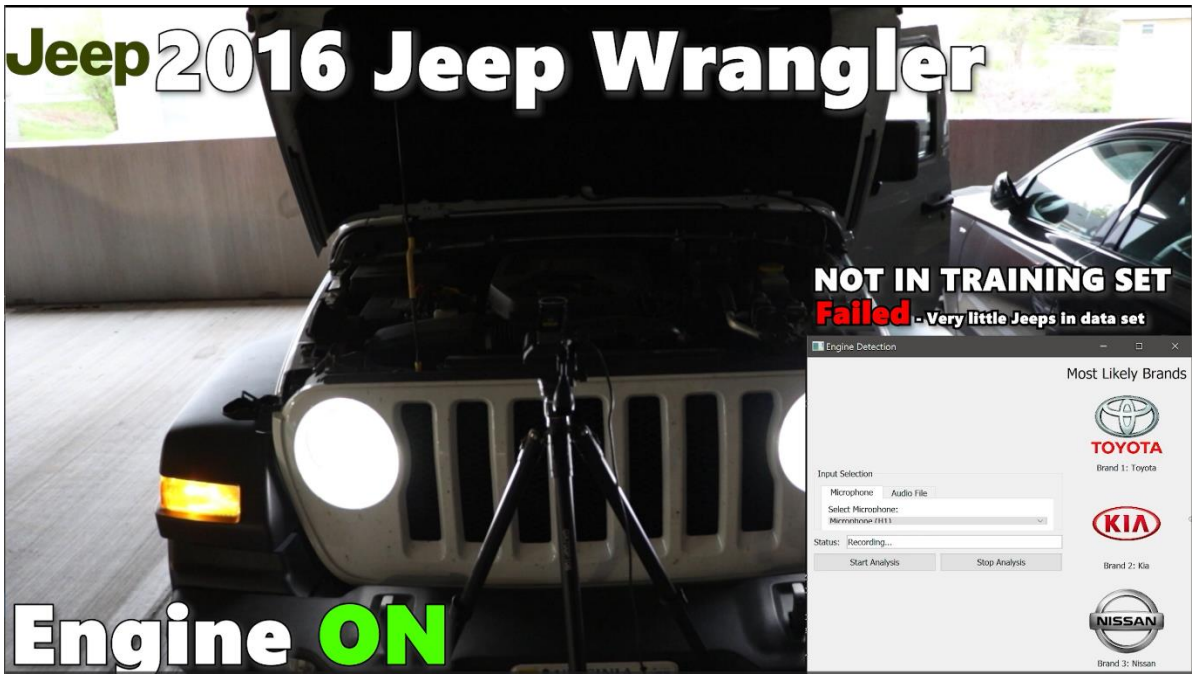
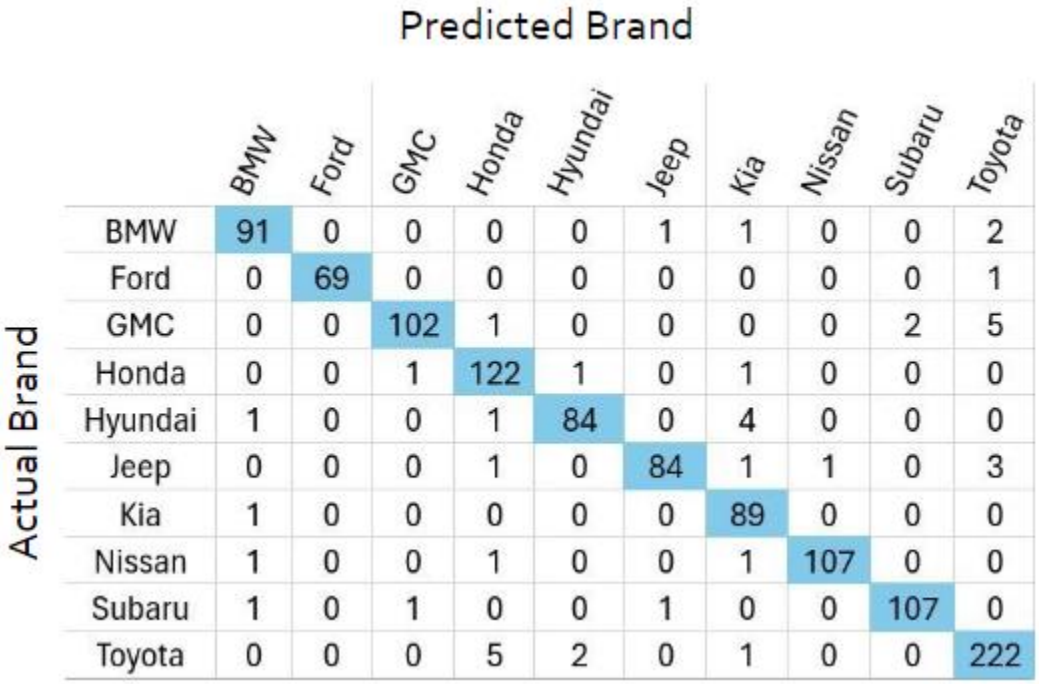


Figure 22: Screen Shot from Demo Video Showing the Model Doesn't Know What Brand Car This is Because the Car Was Not Used in the Testing/Training Set

Despite this, the engine classifier was able to reliably classify audio that was similar to what was included in the dataset. For example, it correctly predicted the brand for all 5 unseen audio clips of cars that were already in the dataset. We also performed a test where each audio clip from the training set went through our data augmentation script with random parameters, creating audio clips that were slightly different from what the model

was trained on. In this test, the model was able to predict the correct car brand 96% of the time. The confusion matrix from this test is shown below.



Actual Brand \ Predicted Brand	BMW	Ford	GMC	Honda	Hyundai	Jeep	Kia	Nissan	Subaru	Toyota
BMW	91	0	0	0	0	1	1	0	0	2
Ford	0	69	0	0	0	0	0	0	0	1
GMC	0	0	102	1	0	0	0	0	2	5
Honda	0	0	1	122	1	0	1	0	0	0
Hyundai	1	0	0	1	84	0	4	0	0	0
Jeep	0	0	0	1	0	84	1	1	0	3
Kia	1	0	0	0	0	0	89	0	0	0
Nissan	1	0	0	1	0	0	1	107	0	0
Subaru	1	0	1	0	0	1	0	0	107	0
Toyota	0	0	0	5	2	0	1	0	0	222

Figure 23: Confusion matrix for engine classifier test with augmented data

Delivery

Overall Project Performance

The last thing to test was the performance of our program as a whole. First, the program is controlled with a user-friendly graphical user interface. It can take input from either a microphone or an audio file, greatly increasing its flexibility and ease of use. In addition, it shows the top 3 predicted car brands in the GUI in 8.5 seconds on average after starting the analysis. The GUI also remains responsive throughout the analysis, allowing the user to stop and change audio input settings at any time.

The program can reliably predict the brand that engine audio belongs to if there are enough audio files for that brand in the dataset and there isn't a high variance between the audio in the dataset and what it's being tested on. For example, the program could accurately classify unseen audio from cars that were already in the dataset, but not new cars as found during outdoor testing. Lastly, the program reliably starts engine classification when an engine is detected in the audio.

Customer Satisfaction

We have not yet received feedback from the customers regarding their satisfaction with the project.

Deliverable Status

The code and documentation for this project have not been given to the customer yet.

Challenges

There were several challenges faced during the development of this project, outlined below.

1) Data collection

Our initial plans for this project were to obtain all the audio training data ourselves by recording our own and other people's cars using high-quality Zoom H1 microphones. However, this approach turned out to be unfeasible, and we could only record audio of about 50 cars on our own. To fix this, friends and family members assisted in the data collection by recording their cars, with both high-quality microphones and those in their cell phones. But even then, this still only increased our recording set by 15 cars.

2) Machine learning model performance

We were unable to collect enough data for the model to be able to fully learn what makes a brand, a brand. In the future we would need to have recorded every single model and year for each make to be able for the model to fully determine what makes a brand sound like that brand. There was no determining feature that the model was able to find to be able to know which brand it was. Thus, our machine learning model was extremely overfitted to the cars we trained it on and would be able to "remember" that car from training, but it failed when given any new data.

3) Model preparation

Another Challenge that we had to address was the size of the dataset on training time. So, we ended up going with an MFCC to train out data on, but this representation of the audio had a lot of compression. For each MFCC sample, we only had 13 different numbers to go off of, and for each 1s clip, we only had 94 MFCC which we then averaged. Instead, we could have looked at a more detailed representation of the audio if we passed the entire spectrogram to the model. Instead of seeing the average of the 13 MFCC per 0.1-

second audio samples, it instead would look at an image of a spectrogram practically doing image classification. Spectrograms are good for more fine detail and thus is why they are used for sound event detection and capture more nuances in audio, while MFCCs are used for limited computational resources. But because a spectrogram is an MxN size, we would have a much higher-dimensional input put into each model. This would cause us to have our model have an input dimensionality of 3D and size of (number of 0.1s audio samples, Height of spectrogram, Width of spectrogram) compared to our current 2D input dimensionality of (977880, 130). With this higher dimensionality, we could instead use a Convolution Neural Network (CNN) which is good at image classification. Using CNNs and spectrograms provides the added advantage of analyzing sequential data over time. When using MFCCs, the temporal information of the audio is lost, and the individual MFCC coefficients are more independent from each other. This can result in the loss of important audio features, such as the rhythmic patterns of an idling engine. We attempted to do this with the spectrogram, instead of MFCC, but even with a Graphics Processing Unit (GPU) containing 12Gb of Graphics Double Data Rate (GDDR) memory, many data points kept running out of memory space and it would take forever to train. So in the future, we could use something like the Hokie Speed Super Computer on campus or the CS Rlogin compute power.

Schedule

Key Milestones:

- Completed initial design concept – Sept. 15, 2023
- Presented preliminary design review – Oct. 26, 2023
- Completed verification test plan – Nov. 12, 2023
- Completed demo of prototype engine classifier with 4 car brands - Nov. 28, 2023
- Presented critical design review – Dec. 1, 2023
- Finished detailed design documentation – Dec. 2, 2023
- Presented prototype demo of project software – Mar. 12, 2024
- Completed engine classifier program – Apr. 5, 2024
- Completed engine detector program - Apr. 10, 2024

Cost Status

As this project is software-based and there is no hardware associated with the deliverables, the cost is $0. The only hardware needed during this project's development

was Zoom H1 microphones for recording car engines. We were able to rent these microphones from the Studio Technologies Lending Desk at Virginia Tech.

Lessons Learned

This project allowed us to learn some important lessons related to working on a long-term project based on machine learning.

1) Data collection

First, we learned that collecting all the data for machine learning models ourselves is difficult and time-consuming. In the previous semester, we planned to collect all our data using Zoom H1 microphones to ensure all audio was high quality. However, we were unable to get enough audio using this approach. So, we had to find audio from various sources online.

2) Machine learning model performance

The main lesson we learned regarding the performance of our models is that without lots of data, machine learning models can very quickly become overfitted. Despite this, there are many ways to decrease the likelihood of overfitting, such as:

- Reduce input dimensionality
- Increase dataset size
- Decrease the number of training epochs
- Add dropout layers
- Normalize/scale input data

3) Teamwork

Last, we learned that working in a team on a long-term project can be challenging. The most important thing to have is open communication and a clearly outlined schedule. This way, everyone knows what each team member is doing and what tasks need to be accomplished next. In addition, it's important to have a good way to share files, especially if the project involves large datasets.

References

[1] M. Perna et al., "Comparison of Road Noise Policies across Australia, Europe, and North America," International Journal of Environmental Research and Public Health, vol. 19, no. 1, pp. 173–173, Dec. 2021, doi: https://doi.org/10.3390/ijerph19010173.

[2] US EPA, “Clean Air Act Title IV - Noise Pollution,” www.epa.gov, Jun. 03, 2015. https://www.epa.gov/clean-air-act-overview/clean-air-act-title-iv-noise-pollution#:~:text=Noise%20pollution%20adversely%20affects%20the,sleep%20disruption%2C%20and%20lost%20productivity

[3] IEEE, “IEEE Code of Ethics,” *ieee.org*, 2020. https://www.ieee.org/about/corporate/governance/p7-8.html (accessed Sep. 22, 2023).