Machine Learning for the Classification of Audio Signals

Course Master of Engineering in Information Technology

Module Individual Project

Gurunag Sai Udaykumar
Mat No: 1387227
gurunag.udaykumar@stud.fra-uas.de

***Abstract*—In a variety of fields, machine learning systems have demonstrated exceptional performance for the classification of datasets. To classify audio data with different frequencies, machine learning techniques have been utilized in this study. The code demonstrates how to take features out of audio files and use supervised learning to categories them into three classes based on material/color: silver, black, and red Algorithms such as MFCC (Mel-frequency Cepstral Coefficients) and Spectrogram are used to preprocess the audio signals to extract frequencies and energies. These audio features are used in Machine Learning algorithms using Support Vector Machines (SVM) and Convolutional neural network (CNN) to identify the audio signals and henceforth generating confusion matrix to determine the model statistics.**

Keywords—Machine Learning, MFCC (Mel Frequency Cepstral Coefficients), Spectrogram CNN (Convolutional neural network), Support vector Machine, Confusion Matrix

I. INTRODUCTION

Audio datasets of the materials are being collected using ASRS (Automated Storage and Retrieval System) using PLC (Programmable Logic Circuit) which automatically picks and places it at three different layers of slab and produces different frequencies of noise when placed on the slabs. The proposed idea provides both monitoring and control of all the parameters using PLC. Datasets around 20,000 have been collected by capturing the audio signals using the Audacity Software. These are categorized based on color of the material as black, red, and silver respectively.

The project aims to design a learning experiment capable of predicting the type of material from the collected audio signals. The Automated Storage and Retrieval System provides a list of user benefits. It includes in saving the labor cost, high floor space utilization, increased safety, and stock rotation. By adding Machine Learning algorithms [1], further add identification of the materials and assists in sorting the material based on the confusion Matrix.

II. METHODOLOGY

Using the existing the system, collect the audio signals using Audacity software with a help of a mic. The mic used is a top end TIE which allows you to make recordings quickly. The audio signals are a collection of the noise produced by the materials. The audio signals are collected at different scenarios where the noises are also recorded along with the input. Using machine learning algorithms, the audio signals are classified and identified based on their frequencies or energies exerted by the materials.

A. Automated Storage and Retrieval System (ARAS)

An Automated Storage and Retrieval System (ASRS) is described by the Material Handling Institute as a set of tools and controls that manages, stores, and retrieves goods within a predetermined degree of operation with accuracy, speed, and precision. The storage system is a three-layered slab where the materials are placed randomly and picked by the Robotic arm and placed at the star shaped receiver. Once the materials are placed, the arm receives the materials and places back in the slab. The PLC (Programmable Logic Controller) is used to pick and drop the materials. Figure 1 is the ASRS used to generate the audio data for the classification.

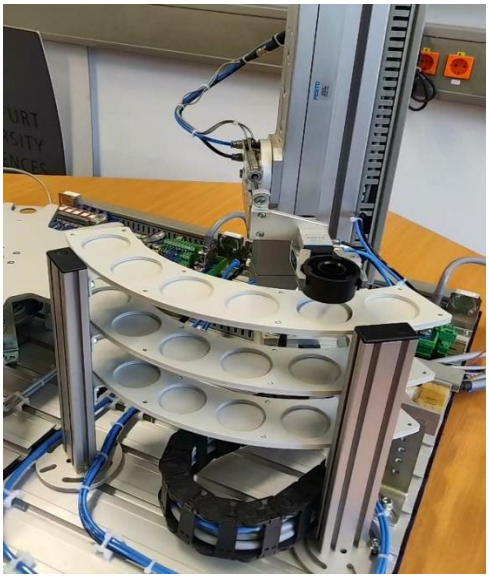

Fig.1.Automated Storage and Retrieval System using PLC

B. Microphone Specification

The microphone used is TIE Studio TUR88 [2] USB Desktop Microphone. The Tie Microphone Pro allows you to make recordings quickly and easily, whilst also functioning as a high-quality output for your headphones to minimize latency to 0 while recording. With its 3x capsules and 4x pickup patterns (cardioid, omni-directional, figure-of-8, and stereo), the TUR88 offers a multitude of recording options, including podcasts, voice-over, and music. Its integrated stereo mode also enables it to function as a room mic for live demos or source separation.

The main features of the TIE studio TUR88 microphone as shown in the figure 2 includes:

- 3x 16mm Electret Microphone Capsules
- 4x Polar Patterns (Cardioid, Omni-Directional, Figure-of-8 & Stereo)
- Frequency Response: 30Hz–20khz (Microphone) & 20Hz-20kHz (Headphone Output)
- Sampling Rate: 96kHz
- Bit Depth: 24-bit
- Dimensions (HxWxD): 280 x 120 x 120mm
- Weight: 1.38kg

Fig.1. TIE studio TUR88 microphone

C. *Mel Frequency Cepstral Coefficients (MFCC)*

MFCCs [3] [4] are derived from the short-term power spectrum of an audio signal after it has been pre-processed with the Mel filter bank. MFCCs capture information about the spectral content of the signal, but with a frequency scale that approximates the human auditory system's perception of sound. MFCC is widely used for speech and audio processing tasks.

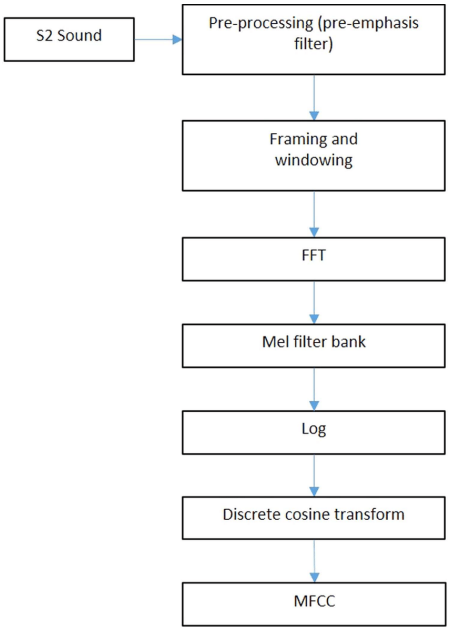

Fig.3. Flowchart of MFCC

The above flow chart in the Figure 3 describes how the MFCC to extract the frequency data from the input audio signal as signal preprocessing. It is particularly effective for tasks like speech recognition, speaker identification, and various audio analysis tasks. After computing the Mel-frequency filter bank energies, the coefficients are transformed using a discrete cosine transform (DCT) to obtain cepstral coefficients. These coefficients capture the spectral characteristics of the audio signal. The features of the audio signals are extracted using the Librosa library which is used for audio signal preprocessing. The features are saved in excel file.

D. *Mel Spectrogram*

A spectrogram [5] is a visual description of audio, which represent time, frequency, and amplitude all in a single graph. A sound spectrogram is also called a Sonogram. Any kind of noise in the audio may be readily identified and isolated with the use of a spectrogram. Spectrograms may be used to identify various vibration kinds by analyzing the frequency content of a waveform.

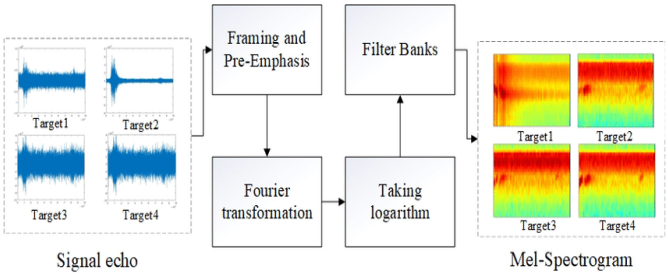

Fig.4. Flowchart of Mel Spectrogram

Figure 4 describes the flow diagram of the Mel-Spectrogram [3] used for preprocessing the audio data for CNN model. Users may utilize the data to find strong signals and track frequency changes over time. A waterfall display is created when data is plotted in three dimensions. Spectrograms are 2D graphs where colors indicate 3D. Time is represented by the horizontal axis, while frequency is represented by the vertical axis. The spectrogram provides a lot of information about the acoustic elements of sound.

A machine learning classifier can be utilised in conjunction with the output spectrogram picture. Several learning algorithms, spectrogram analysis, and feature extraction may be applied to carry out fundamental classifications that offer more in-depth understanding of the audio. Most audio clips also include several unique peculiarities in their spectrograms. These spectrogram characteristics aid in the effective classification of audio.

E. *Machine Learning*

Machine learning is an artificial intelligence subject that focuses on creating algorithms and models that allow computers to learn from data and make predictions or judgments without being explicitly programmed. The core concept of machine learning is to allow computers to learn from examples and experience rather than depending on explicit rules or human involvement.

Algorithms and models in machine learning are trained using a dataset that contains input data and the matching output values. The goal is to train a function to predict output values for new, previously unknown input data. The learning process entails iteratively modifying the model parameters to reduce the discrepancy between the predicted and true outputs.

There are several types of machine-learning algorithms, including supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithm is trained on a labelled dataset where the input data is paired with the correct output values. In unsupervised learning, the algorithm is trained on an unlabeled dataset, and the goal is to discover patterns and structure in the data. To analyze the experiment, Supervised learning is used, where two different models are created with different algorithms [1], the Convolutional Neural Network (CNN) algorithm and Support Vector Machine (SVM) Algorithm respectively

F. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [6] are a type of neural network architecture used in image and video recognition. CNNs are intended to detect and extract characteristics from input image, audio, or video data to classify or recognize objects in the photos, audio, or videos.

A CNN's architecture, as shown in figure 5, often consists of multiple layers, such as convolutional layers, pooling layers, and fully linked layers. Filters or kernels are applied to the input image in the convolutional layers to recognize characteristics like as edges, lines, and textures. The pooling layers are used to minimize the spatial dimensionality of the output from the convolutional layers, which aids in lowering the network's computational complexity. Finally, the fully connected layers are used to classify or recognize the objects in the image or video.

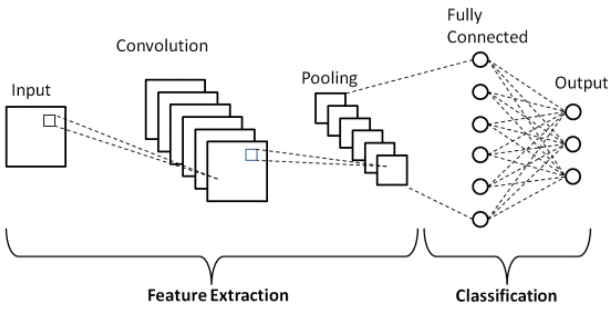

Fig.5. Convolutional Neural Network Architecture

The following steps are described to create a CNN model to detect a person wearing winterwear inside the car

- Data Preparation: The first step is to prepare the dataset for training. This involves splitting the dataset into training, validation, and test sets, and pre-processing the images by resizing, normalizing, and augmenting them.
- Building the CNN: The CNN architecture is designed by specifying the number of convolutional layers, pooling layers, and fully connected layers. The number of filters and kernel sizes are also specified. The input and output dimensions of each layer are calculated based on the size of the input images and the number of output classes.
- Compiling the Model: After building the CNN architecture, the next step is to compile the model. This involves specifying the loss function, optimizer, and metrics to be used during training.
- Training the Model: The model is trained on the training set by feeding batches of images into the network and adjusting the weights based on the error between the predicted output and the true output. The training process continues until the model achieves a satisfactory level of accuracy on the validation set.
- Evaluating the Model: After training the model, it is evaluated on the test set to measure its performance on unseen data. The accuracy, precision, recall, and F1 score are commonly used metrics for evaluating the performance of a CNN.
- Fine-tuning the Model: If the performance of the model is not satisfactory, it can be fine-tuned by adjusting the hyperparameters such as the learning rate, batch size, and number of epochs. The architecture of the CNN can also be modified by adding or removing layers.
- Predicting on Test Data: Once the CNN is trained and evaluated, it can be used to predict the class of new images. The input image is fed into the network, and the output is the predicted class label.

G. Support Vector Machines (SVM)

Support Vector Machine (SVM) [7] is a supervised machine learning algorithm used for classification, regression, and outlier detection. The algorithm tries to find the best possible decision boundary that separates the data points into different classes. SVM algorithm works by transforming the data into a higher dimensional space, where it finds the optimal boundary that separates the data points.

There are two types of SVM: linear SVM and nonlinear SVM. Linear SVM tries to find the best hyperplane that separates the data points into different classes, where Non-linear SVM tries to find the best decision boundary by transforming the data into a higher dimensional space using a kernel function which is described in the figure 6.

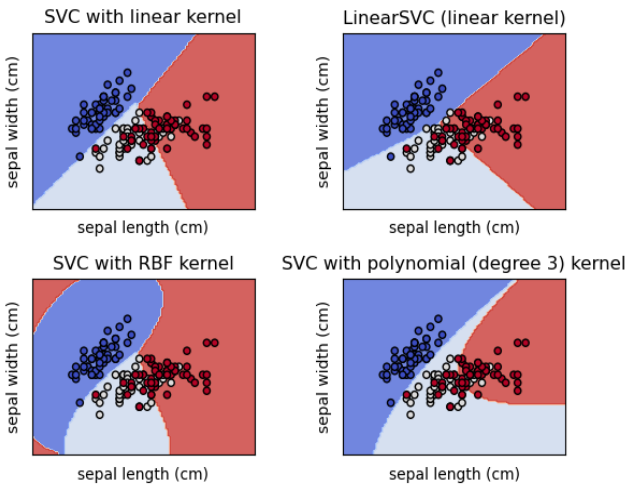

Fig.6. SVC with linear and non-linear Kernel

Like the steps of the CNN model, The SVM model has data pre-processing, Creation, Evaluation, and tuning phase, which are discussed below.

- Prepare the data: Load the dataset and split it into training and testing sets. It is important to pre-process the data, including scaling and normalization, as SVM is sensitive to the scale of the features.
- Create an instance of the SVM model: In scikit-learn, you can create a linear SVM model using the Linear SVC class. Set the hyperparameters, such as the regularization parameter C, which controls the trade-off between maximizing the margin and minimizing the classification error.
- Train the model: Fit the SVM model to the training data using the fit() method.
- Test the model: Predict the labels of the test data using the predict() method and evaluate the model's performance using metrics such as accuracy, precision, and recall.

H. Confusion Matrix

By comparing the anticipated labels of a collection of data to the actual labels, a confusion matrix is a table that summarizes the performance of a classification method. It is a commonly used method in statistics and machine learning to assess the precision of a classification model.

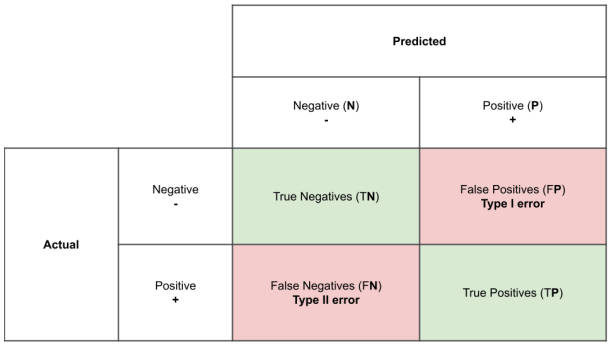

Fig.7. confusion matrix

The above figure 7 consists of several parameters which are calculated based on four core parameters:

True Positive (TP): When the expected value matches the actual value, which determines the true positive case. This indicates that the model predicted a positive result, and the actual result is positive.
True Negative (TN): When the expected value and the real value are the same, a truly negative situation arises. This time, however, the model predicted a negative value, and the actual value is also negative.
False Positive (FP): When the expected value matches the incorrect value, this is known as a false positive. The model anticipated a favorable outcome even though the actual value was negative. This is called the first type of prediction error.
False Negative (FN): When the expected value matches the incorrect value, this is referred to as a false negative case. Even though the actual number was positive, the model predicted a negative result. This is the second type of error in the prediction model, and it assists us in obtaining additional parameters.

From the confusion matrix, several important metrics can be calculated to evaluate the performance of the classification algorithm as mentioned below:

Accuracy: Accuracy is the ratio of the correctly predicted samples to the total number of samples in the dataset. It measures how often the classifier makes the correct prediction. The formula for accuracy is:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Precision: Precision is the ratio of the correctly predicted positive samples to the total number of positive samples predicted by the model. It measures how many of the predicted positive samples are positive. The formula for precision is:

$$\text{Precision} = TP / (TP + FP)$$

Recall (Sensitivity): Recall is the ratio of the correctly predicted positive samples to the total number of positive samples in the dataset. It measures how many of the actual positive samples were correctly predicted by the model. The formula for recall is:

$$\text{Recall} = TP / (TP + FN)$$

Specificity: Specificity is the ratio of the correctly predicted negative samples to the total number of negative samples in the dataset. It measures how many of the actual negative samples were correctly predicted by the model. The formula for specificity is:

$$\text{Specificity} = TN / (TN + FP)$$

F1 Score: F1 score is the harmonic mean of precision and recall. It is a good metric when both precision and recall are important. The formula for F1 score is:

$$\text{F1 Score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

The confusion matrix is a table used to evaluate the performance of a classification model. It summarizes the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) that a model produces when predicting the class of a set of data. With the aid of these numbers, which may determine a number of rates or metrics, such as the true positive rate (TPR), false positive rate (FPR), true negative rate (TNR), and false negative rate (FNR), which are calculated as follows:

- True Positive Rate (TPR), also called Sensitivity or Recall: The proportion of true positives over the total number of actual positives. It is calculated as TP / (TP + FN).
- False Positive Rate (FPR): The proportion of false positives over the total number of actual negatives. It is calculated as FP / (TN + FP).
- True Negative Rate (TNR), also called Specificity: The proportion of true negatives over the total number of actual negatives. It is calculated as TN / (TN + FP).

- False Negative Rate (FNR): The proportion of false negatives over the total number of actual positives. It is calculated as FN / (TP + FN).

These rates are useful for evaluating various aspects of a classification model's performance, depending on the problem and application. TPR measures the model's capacity to accurately identify positive examples, whereas FPR measures the model's tendency to wrongly classify negative situations as positive. TNR is a measure of the model's ability to correctly identify negative situations, whereas FNR is a measure of the model's tendency to wrongly classify positive cases as negative. In general, to maximize TPR and TNR while minimizing FPR and FNR to build a high-performing classification model.

III. IMPLEMENTATION

The Implementation of the project is accomplished using Python as programming language which offers support to many libraries for Machine learning, Statistics and Computations. The implementation to achieve the goal of the project is discussed below.

A. Data Collection

For a period of more than three months, collection of the dataset using the hardware i.e., ASRS using PLC to collect the data samples. Three different cylindrical materials are randomly placed in the slab. The robotic hand picks the cylindrical material one by one and places on the star shaped rotating place. This is an automated process where once a complete rotation is finished robotic hand picks the material and places it back in the slab as shown in the figure 8.

The current operation is such that the robotic arm just picks the material and places back at the slab. When the materials are placed on the slab or the rotating star shaped slab a noise is produced. The three different materials which can identified by color in vision which is 'Black,' 'Red' and 'Silver.' They also produce different noise which is very evident by hearing.

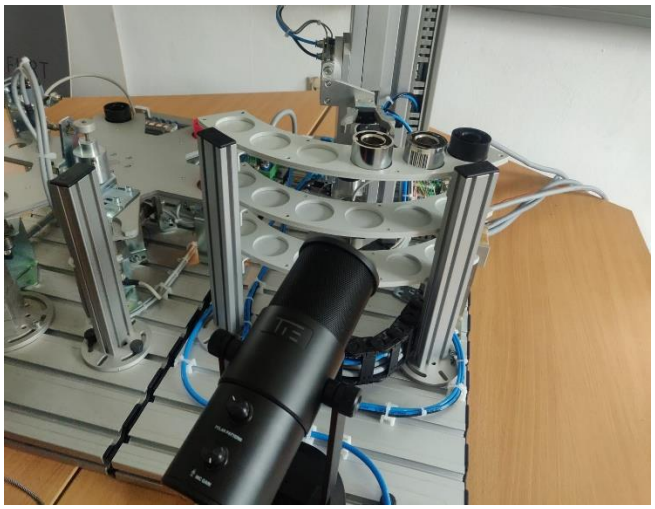

Fig.8. TIE microphone setup to get the audio data from ASRS

Using programming language, how this can be differentiated by sound and identify each of it with good accuracy. There are also surrounding noises when the ASRS is running. Hence, also added few scenarios can be used with the surrounding noise along with the material sound which can be used to reduce or eliminate the surrounding noise and predict the noise produced by the material. The scenarios that are considered for the collection of data are mentioned as below:

- Data samples without any background noise
- Data samples with environmental noises like sounds produced by ventilator, our outside vehicle honks, noises created by nearby machines etc.
- Data samples with artificially created noise like playing videos/music from phone

TABLE.1. COLLECTION OF TRAINING DATA

Class	Class label	Number of Data
Black	1	6000
Red	2	6000
Silver	3	6000
Total		**18000**

Based on the mentioned scenarios, as per the table 1, A total of 18000 datasets are collected where 6000 datasets are collected for each class object which consist of all the acoustic scenarios of collecting data as mentioned above.

B. Python Libraries Used

Python offers a few libraries to support the Machine learning programming and the following libraries are used to reach the goal of the project.

- Librosa: Librosa is a Python package for analyzing and processing audio signals, used for feature extraction and display of the audio data.
- NumPy: NumPy is a powerful numerical computing library for Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
- Pandas: Pandas is an open-source data manipulation and analysis library for Python. It provides data structures for efficiently storing and manipulating large datasets, along with tools for reading and writing various data formats.
- Scikit-Learn: sklearn provides simple and efficient tools for data mining and data analysis, built on top of other scientific computing libraries such as NumPy, SciPy, and Matplotlib.
- Keras: Keras is designed to be user-friendly, modular, and extensible, allowing for easy and quick prototyping of deep learning models. Keras acts as an interface for the TensorFlow and Theano libraries, which are popular deep learning frameworks

C. Data Labeling and Extraction

Traditionally, data has been manually labelled, a labor- and resource-intensive process. On the other hand, by initially learning them on a portion of manually labelled data, machine learning models or algorithms may be used to auto-label data.

In the section, a Target value column as per the use case and drop the unwanted column and labeled the data as per required. The below figure 9 represents a single audio data file plotted against the amplitude vs time graph.

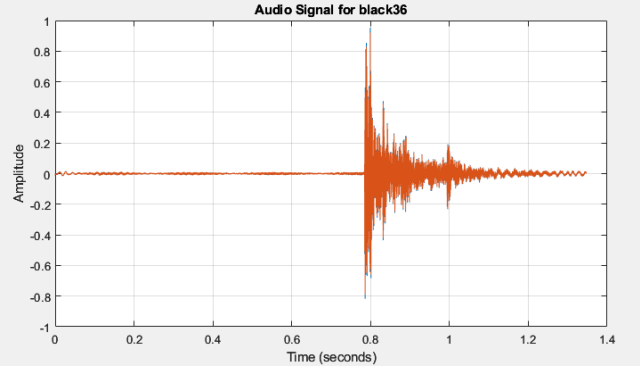

Fig.9. Audio data plot (Amplitude vs time graph)

The below code snippet in the figure 10 describes the method to retrieve the training data filed form the csv file as follows. The method 'get_traindata' orchestrates the conversion of training audio data into a suitable format by loading information from a designated CSV file ('TrainDataSetCSV.csv') into a Pandas Data Frame. Following this, it verifies the existence of a directory labeled 'train_extracted' within the specified file path and generates it if absent, serving as the repository for the resultant NumPy arrays. Extracting crucial details like file names and relative file paths from the Data Frame, the process systematically accesses each audio file in the dataset.

```
# Function to extract the training data and convert to numpy files
def get_traindata(dir_path,path_traindata):

    # Load the training data csv file into a dataframe.
    df = pd.read_csv(os.path.join(dir_path,'TrainDataSetCSV.csv'))

    # Creating folder to store the Numpy arrays if they don't exist.
    if not os.path.exists(os.path.join(dir_path,'train_extracted')):
        os.makedirs(os.path.join(dir_path,'train_extracted'))

    # Getting the file names of audios from the dataframe.
    audio_files = np.array(df['file name'])
    audio_files_path = np.array(df['Relative file path'])

    # Load each audio file, save it as a numpy array
    for i in range(len(audio_files)):
        path = os.path.join(dir_path, path_traindata, str(audio_files_path[i]).zfill(8))
        d,r = librosa.load(path)
        np.save(os.path.join(dir_path, 'train_extracted',str(audio_files[i].replace(".wav",""))+'.npy'),d)
```

Fig.10. Code Snippet to retrieve Training Dataset

Employing 'librosa.load', it procures the audio waveform and its sampling rate, subsequently preserving this data as NumPy arrays within the 'train_extracted' directory. Each audio file undergoes transformation, being saved as a NumPy array named after its original file name but with the '.npy' extension appended. This systematic approach allows for the organization and pre-processing of audio data, rendering it conducive for subsequent machine learning model training or any pertinent audio-centric analyses.

D. Data Preprocessing

1) Mel Frequency Cepstral Coefficients (MFCC)

The get_mfcc_features function extracts Mel-Frequency Cepstral Coefficients (MFCC) features from audio sequences in each directory. The function accepts three parameters: path_to_train, csv_file, and extracted_folder. path_to_train is the directory path where the training data is placed, csv_file is the name of the CSV file containing information about the audio files, and extracted_folder is the folder where pre-processed audio data in NumPy format is stored.

The approach begins by using pandas to import the CSV file into a Data Frame. The names of the audio files are then retrieved from the Data Frame's 'file name' column. The retrieved features are then stored in an empty list called mfcc_features.

The method loads the relevant pre-processed audio sequence in NumPy format from the supplied folder for each audio file. The MFCC coefficients are then calculated using the librosa package. For each set of MFCC coefficients, several statistical metrics such as mean, median, standard deviation, skewness, kurtosis, maximum, and minimum values are produced. These statistics are combined into a single array and added to the list of mfcc_features.

The function loops over all audio files, extracting and aggregating MFCC characteristics as it goes. Finally, the list of MFCC features is returned, offering a full collection of acoustic characteristics for training machine learning models on audio data in the supplied directory.

2) Standardizing MFCC features

Standardizing features, especially Mel-Frequency Cepstral Coefficients (MFCCs), is crucial in the context of machine learning applications, particularly when working with audio signal processing and speech recognition. MFCCs, derived from various operations on audio signals, often exhibit different scales. Standardization addresses this issue by ensuring uniformity in feature scales. This practice is particularly beneficial for machine learning algorithms sensitive to feature scales for Support Vector Machines. The below Figure 11 is the python snippet for implementation of standardizing the input audio data features after MFCC preprocessing.

The standardize_features function is used to standardize feature matrices in datasets used for machine learning. To make sure that features have a consistent scale which is a critical preprocessing step for many machine learning algorithms. The function uses the scikit-learn StandardScaler.

The training and testing datasets are represented by the two feature matrices, X_train and X_test, which are supplied as inputs to this function. The option with_mean=False is used to initialize the StandardScaler, indicating that the data will not be centered around zero during the standardization process.

```
# Funtion to standardize the features
def standardize_features(X_train,X_test):

  # Initialize standard scalar with zero mean
  sc = StandardScaler(with_mean=False)
  # Fit and transform the Training Dataset.
  X_train= sc.fit_transform(X_train)
  # Transform the testing set.
  X_test = sc.transform(X_test)

  return X_train,X_test
```

Fig.11. Method to Standardize input data features

The function then applies the fit_transform method to transform the training dataset. This method calculates the training data's mean and standard deviation and standardizes it according to these statistics. The same mean and standard deviation that were calculated from the training data are then applied to the testing dataset. By doing this, homogeneity in feature scales is maintained and a consistent standardization approach is applied to the training and testing datasets.

3) Spectrogram

The get_spectrograms method extracts Mel spectrogram features from audio sequences for training and testing datasets. Six arguments are required by the function: path_to_dir specifies the directory path in which the data is stored. train_csv is the name of the CSV file that contains information about the training audio files, train_extracted is the folder that contains pre-processed training audio data in NumPy format, test_csv is the name of the CSV file that contains information about the testing audio files, test_extracted is the folder that contains pre-processed testing audio data, and target_shape specifies the desired shape for the spectrograms (defaulting to (128, 128)).

To store the training spectrograms, training labels, and testing spectrograms, respectively, the procedure initializes empty lists called X_train, Y_train, and X_test. To resize or pad the spectrograms to the desired target form, it additionally offers a nested function called resize_spectrogram.

The process then iterates over the training set, loading every audio clip, using librosa to compute the Mel spectrogram, and then resizing and padding each one to fit the desired form. The associated label is added to Y_train, and the resultant spectrogram is appended to X_train. To add spectrograms to the X_test list, follow the same procedure with the testing data.

Following data processing, NumPy arrays are created from the lists. Furthermore, the technique maintains the class distribution by dividing the training features into training and validation sets using StratifiedShuffleSplit from scikit-learn.

In the end, the spectrogram features and matching labels for training and validation are provided by the NumPy arrays training_data, train_labels, val_data, and val_labels. When producing spectrogram data for machine learning model training on audio categorization problems, this approach might be helpful. The below figure 11 is a sample spectrogram generated from the input audio signal related to the figure 9.

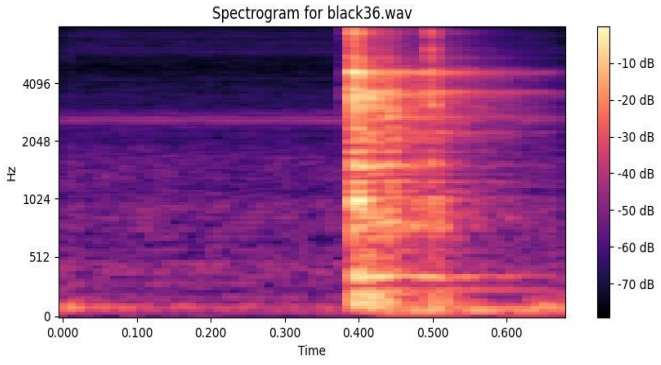

Fig.11. Spectrogram representation of a single audio data of figure 9

E. SVM implementation with MFCC

The Figure 12 describes the code snippet for the SVM model for the MFCC input audio data. Using the one-vs-rest (OvR) technique, the svm_classifier method constructs a Support Vector Machine (SVM) classifier for multiclass classification. Because of its versatility in handling non-linear decision boundaries, the Radial Basis Function (RBF) kernel is a widely used option for Support Vector Machines (SVMs).

```
# Function to use Support Vector Machines classifier
def svm_classifier(X_train,Y_train,X_test):

  # Intialize SVM classifier with One-vs-Rest decision function.
  svm_model = svm.SVC(kernel='rbf', decision_function_shape='ovr')
  # Fit the Training Dataset.
  svm_model.fit(X_train, Y_train)
  # Predict labels for the test dataset.
  Y_test = svm_model.predict(X_test)

  return Y_test
```

Fig.12. SVM Classifier

Fitting the SVM model to the given training dataset (X_train and Y_train) is the task of the training phase. The training set's feature matrix, which contains the input data, is represented by X_train, and the labels that go with it are stored in Y_train. The SVM model gains the ability to draw decision boundaries in the feature space between various classes as it trains.

The SVM model is used to predict labels for a different test dataset (X_test) after it has been trained. The variable Y_test contains the expected labels. The SVM model is predict technique classifies test set instances using the learned decision boundaries.

F. CNN implementation with Spectrogram

The code snippet in the Figure 13 describes the CNN model implemented for the spectrogram input data. Using the `get_spectrograms` function, the spectrogram data is first divided into training, validation, and test sets in this script. Next, using the `cnn_classifier` function, it builds a Convolutional Neural Network (CNN) model and specifies its architecture for spectrogram analysis.

```
#Function to use CNN classifier
def cnn_classifier(inputShape):
    # Intializing the model sequential.
    model = Sequential()

    # Convolutional layers
    model.add(layers.Conv2D(16, (4, 4), activation='relu', input_shape=inputShape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.BatchNormalization())

    model.add(layers.Conv2D(32, (4, 4), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.BatchNormalization())

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4,activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    print(model.summary())

    return model
```

Fig.13. CNN Classifier

Convolutional layers are critical in extracting hierarchical characteristics from input data. The first convolutional layer implements the Rectified Linear Unit (ReLU) activation function and contains 16 filters of size (4, 4). Following that, a max-pooling layer with a (2, 2) window is used to reduce spatial dimensions and improve computing efficiency. Batch normalization is intentionally placed after each pooling layer to normalize activations, contributing to the training process's stability.

The second convolutional layer expands on these concepts, with 32 filters of size (4, 4) with ReLU activation and a similar max-pooling and batch normalization structure.

A flatten layer is included to move from the convolutional layers to the highly connected layers, so reducing the three-dimensional output to a one-dimensional format. The model can recognize complex patterns in the learnt features because to the densely connected layers, which include a thick layer with 64 units and ReLU activation. A dropout layer with a 0.5 dropout rate is introduced after the dense layer in order to reduce overfitting.

The model's last layer is devoted to generating classification outcomes. It consists of a four-unit dense layer that uses the SoftMax activation function to create class probabilities in a multi-class classification job. In multi-class classification scenarios, categorical_crossentropy is selected as the suitable loss function, and the model is trained to minimize it.

The model's parameters are iteratively adjusted using the Adam optimizer to maximize the model's performance. The model's accuracy is tracked throughout training as a metric for assessment to determine how well it can categorize images.

IV. RESULTS AND DISCUSSION

A. Extraction of test data

The test data is collected while collection of training data and a total of 2100 audio data is collected across three classes where 700 sets of data collected respectively for each class. As mentioned in the table 2.

TABLE.2. COLLECTION OF TESTING DATA

Class	Actual Class label	Number of Data
Black	1	700
Red	2	700
Silver	3	700
Total		**2100**

Similar method is implemented for converting and saving the test data. The contents of the test data are saved into a csv file ‘TestDataSetCSV.csv’ which has the test audio data access file path, actual class name and label, and audio signal properties The Figure 14 describes the code snippet for implementation of extraction of test data set from the CSV file.

The 'get_labels' function retrieves and returns class labels from a CSV file containing pertinent data. The function accepts two parameters: 'original_path' and 'csv_file'. 'original_path' indicates the directory path where the CSV file is placed, and 'csv_file' is the name of the CSV file. The function reads the CSV file into a Data Frame using the pandas library, assuming the CSV file has a column named 'Class ID' that contains the class labels.

```
# Function to get test data and convert it into numpy files
def get_testdata(dir_path, path_testdata):

    # Load the training data csv file into a datwaframe.
    df = pd.read_csv(os.path.join(dir_path,'TestDataSetCSV.csv'))

    # Creating folder to store the Numpy arrays if they don't exist.
    if not os.path.exists(os.path.join(dir_path,'test_extracted')):
        os.makedirs(os.path.join(dir_path,'test_extracted'))

    # Getting the file names of audios from the dataframe.
    audio_files = np.array(df['file name'])
    audio_files_path = np.array(df['Relative file path'])

    # Load each audio file, save it as a numpy array
    for i in range(len(audio_files)):
        path=os.path.join(dir_path, path_testdata, str(audio_files_path[i]))
        d,r = librosa.load(path)
        np.save(os.path.join(dir_path,'test_extracted',str(audio_files[i].replace(".wav",""))+'.npy'),d)
```

Fig.14. Method to retrieve test Dataset

Following that, the procedure retrieves the 'Class ID' column and transforms it to a NumPy array containing the labels. Finally, the array of labels is delivered, allowing you to easily access and utilize class labels from the requested CSV file in the given directory.

The test data is preprocessed further preprocessed based on the respective training preprocessing method as MFCC for SVM or Spectrogram for CNN respectively and further sent to the prediction method once the model is ready to predict after the training phase is finished.

B. SVM model without standardizing MFCC features

Considering the implementation of the SVM model without any standardizing the features for the MFCC input data. Once the model training is finished, the model is ready for the prediction and the test data as discussed in the table 2 is sent as prediction input to the model and henceforth the Figure 15 represents the confusion matrix for the same and the table 3 represents the metrics of the confusion matrix.

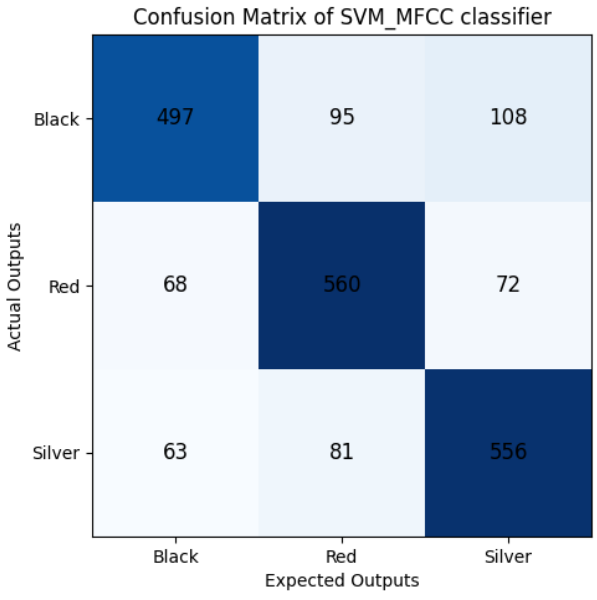

Fig.15. Confusion matrix for the SVM without standardizing the preprocessed MFCC input data.

TABLE.3. CONFUSION MATRIX METRIS FOR THE SVM WITHOUT STANDARDIZING THE PREPROCESSED MFCC INPUT DATA

Parameter	Value in Percentages
Accuracy	76.81%
Precision	76.92%
Recall	76.81%
F1 Score	76.76%
True Positive Rate	[71%, 80%, 79.43%]
True Negative Rate	[90.64%, 87.43%, 87.14%]
False Positive Rate	[9.36%, 12.57%, 12.86%]
False Negative Rate	[29%, 20%, 20.57%]

From the generated confusion matrix by statistical analysis the table 3 represents the parameters which states the accuracy of model is generated. The metrics are obtained as follows with accuracy of 76.81%, precision of 76.92%, recall of 76.81% and 76.76%.

Further the rates of confusion matrix values are also generated, where in the table 3 it is represented in an array format based on the class data i.e. black, red, and silver classes. For the class Black, the true positive rate of 71%, true negative rate of 90.64%, false positive rate of 9.36% and false negative rate of 29% is achieved. Similarly for the class red, the TRP, TNR, FPR and FNR are 80%, 87.43%, 12.57%, and 20% respectively, and for the class silver, 79.43%, 87.14%, 12.86%, and 20.57%.

C. SVM model with standardizing MFCC features

Considering the implementation of the SVM model without any standardizing the features for the MFCC input data. Once the model training is finished, the model is ready for the prediction and the test data as discussed in the table 2 is sent as prediction input to the model and henceforth the Figure 16 represents the confusion matrix for the same and the table 4 represents the metrics of the confusion matrix.

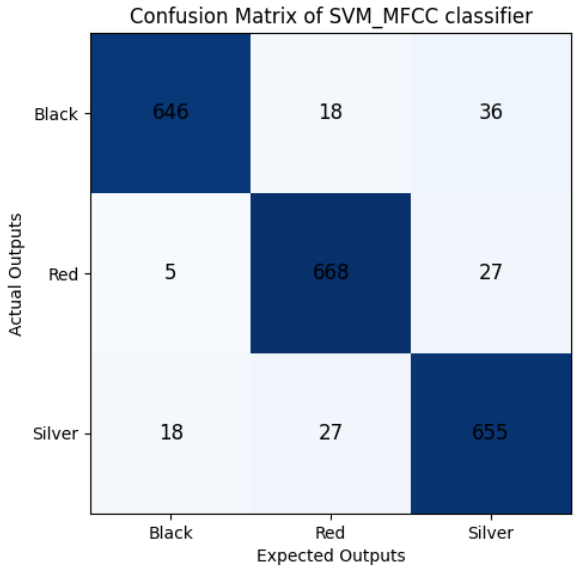

Fig.16. Confusion matrix for the SVM with standardizing the preprocessed MFCC input data.

TABLE.4. CONFUSION MATRIX METRIS FOR THE SVM WITH STANDARDIZING THE PREPROCESSED MFCC INPUT DATA

Parameter	Value in Percentages
Accuracy	93.76%
Precision	93.83%
Recall	93.76%
F1 Score	93.77%
True Positive Rate	[92.29%, 95.43%, 93.57%]
True Negative Rate	[98.36%, 96.79%, 95.5%]
False Positive Rate	[1.64%, 3.21%, 4.5%]
False Negative Rate	[7.71%, 4.57%, 6.43%]

From the generated confusion matrix by statistical analysis the table 4 represents the parameters which states the accuracy of model is generated. The metrics are obtained as follows with accuracy of 93.76%, precision of 93.83%, recall of 93.86% and 93.77%.

Further the rates of confusion matrix values are also generated, where in the table 4 it is represented in an array format based on the class data i.e. Black, Red, and silver classes. For the class Black, the true positive rate of 92.29%, true negative rate of 98.36%, false positive rate of 1.64% and false negative rate of 7.71% is achieved. Similarly for the class red, the TRP, TNR, FPR and FNR are 95.43%, 96.79%, 3.21%, and 4.57% respectively, and for the class silver, 93.57%, 95.5%, 4.5%, and 6.43%.

Comparing the following with the results of the SVM model without standardizing the MFCC features states that having the standardization of the extracted MFCC frequency features results in the better confusion matrix metrics as it ensures the uniformity in the feature scales for the Support Vector Machine model.

D. CNN model with Spectrogram

Considering the implementation of the CNN model with the Spectrogram as the input data for the model, Once the model training is finished, the model is ready for the prediction and the test data as discussed in the table 2 is preprocessed with spectrogram method and sent as prediction input to the model and henceforth the Figure 17 represents the confusion matrix for the same and the table 5 represents the metrics of the confusion matrix.

From the generated confusion matrix by statistical analysis the table 5 represents the parameters which states the accuracy of model is generated. The metrics are obtained as follows with accuracy of 84.71%, precision of 85.44%, recall of 84.71% and 84.89%.

Further the rates of confusion matrix values are also generated, where in the table 5 it is represented in an array format based on the class data i.e. Black, red, and silver classes. For the class Black, the true positive rate of 84.57%, true negative rate of 94.86%, false positive rate of 5.14% and false negative rate of 15.43% is achieved. Similarly for the class red, the TRP, TNR, FPR and FNR are 84.43%, 96.43%, 3.57%, and 15.57% respectively, and for the class silver, 85.14%, 85.79%, 14.21%, and 14.86%.

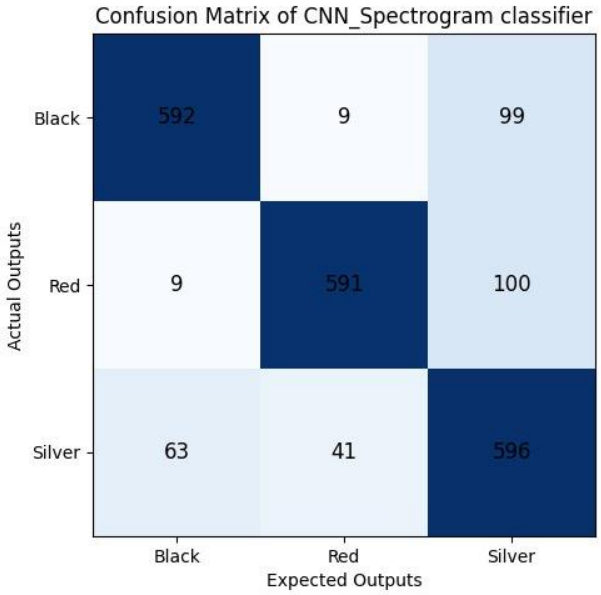

Fig.17. Confusion matrix for the CNN model with spectrogram input.

TABLE.5. CONFUSION MATRIX METRIS FOR THE CNN MODEL WITH SPECTROGRAM INPUT

Parameter	Value in Percentages
Accuracy	84.71 %
Precision	85.44%
Recall	84.71%
F1 Score	84.89%
True Positive Rate	[84.57%, 84.43%, 85.14%]
True Negative Rate	[94.86%, 96.43%, 85.79%]
False Positive Rate	[5.14%, 3.57%, 14.21%]
False Negative Rate	[15.43%, 15.57%, 14.86%]

Comparing the CNN model with the SVM model, the SVM model with the standardized MFCC input data provide an accuracy of 93.76% as of the CNN model with spectrogram of 84.71%. To achieve higher accuracy with CNN model many datasets is required rather to that of a SVM model which yielded the deviation in the confusion matrix parameters between them. Considering the model training and prediction time, SVM model performs way ahead than CNN model, but once the CNN model is trained a possibility to save the trained model exists and henceforth this can be used in the data prediction.

V. CONCLUSION

To distinguish the audio of materials, the machine learning models have been applied and data are collected from the storage system using the TIE microphone. The audio signals provided by the materials have different frequencies and energy. Only three different materials are used and a few more can be added in the future. Using the collected data, SVM and CNN classifiers are implemented to classify the testing data into different classes based on the audio.

With usage of SVM and CNN classifier models, the model is effective in classifying the data. Three different results have been obtained to analyze the efficiency of the models. SVM with Standardization, SVM without Standardization, Spectrogram with CNN.

From the above results, SVM with standardization has performed well or has good accuracy compared to other classifying models. When checking on SVM with no standardization which has also provided good accuracy but can be improved. But comparing these two models, standardization has improved the accuracy of the model. Further CNN with spectrogram is also implemented and comparing it to SVM model there is a deviation in the confusion matrix parameters as the CNN model requires a larger dataset to provide higher accuracy. These models can be provided with more inputs and classes which can be used to train and more test data to predict and improve the model performance.

The classification models in this project have helped the research in this direction even though this may not fully comprehend the features that are contained in the data. Overall increasing the amount of training data is the only surefire approach to raise classification rates. Collective effort can be used by training larger pretrained models with the data from our experiment to distinguish between audio.

VI. REFERENCES

[1] K. Zaman and M. Sah, "A Survey of Audio Classification Using Deep Learning," 2023. [Online]. Available IEEE DOI: 10.1109/ACCESS.2023.3318015.

[2] T. Studio, "TIE Studio TUR88 Specifications," [Online]. Available: https://tie-products.com/wp-content/uploads/2017/12/bedienungsanleitung-TUR88.pdf.

[3] U. Garg and S. Agarwal, "Prediction of Emotions from the Audio Speech Signals using MFCC, MEL and Chroma," 2020. [Online]. Available IEEE DOI: 10.1109/CICN49253.2020.9242635.

[4] D. Trijatmiko and A. Y. Rahman, "Voice Classification of Children with Speech Impairment Using MFCC Kernel-Based SVM," 2023. [Online]. Available IEEE DOI: 10.1109/ICCoSITE57641.2023.10127773.

[5] M. Mulimani and S. G, "Acoustic Event Classification Using Spectrogram Features," 2018. [Online]. Available IEEE DOI: 10.1109/TENCON.2018.8650444.

[6] P. Kalapatapu and J. L. Sravani, "Genre Classification using Spectrograms as input to CNN on Indian Music," 2021. [Online]. Available IEEE DOI: 10.1109/ICETCI51973.2021.9574060.

[7] Hayder Hasan, Helmi Z.M. Shafri and Mohammed Habshi, "A Comparison Between Support Vector Machine (SVM) and Convolutional Neural Network (CNN) Models For Hyperspectral Image Classification," iopscience, 2019. [Online]. Available: DOI: 10.1088/1755-1315/357/1/012035.