Machine Learning Models For Audio Classification

Project Category

Application of Machine Learning to a Practical Problem

CS 5785 - Applied Machine Learning

Cornell Tech | Fall 2023

William Reid Hernandez (wjr83)

Kexin Cheng (kc2248)

Hongjin Quan (hq48)

1. ABSTRACT

This paper presents an exploration of ten machine-learning models to classify audio samples from the TinySOL audio dataset into instrument types and instrument families. The models evaluated in the experiment are VGGish, K-Nearest-Neighbors (KNN), Support Vector Machine (SVM), Logistic Regression, Random Forest, Decision Tree, Gaussian Naive Bayes, Conv1D, Conv2D, and the Long Short-Term Memory (LSTM) model. Techniques for fine-tuning these models and reducing the high dimensionality of features to enhance performance were explored. The outcome of the experiment showed that the three CNN models (Conv1D, Conv2D, and LSTM) outperformed all other models in this classification task of both instrument type and family type. In contrast, the Gaussian Naive Bayes model showed the worst performance before and after relying on Principal Component Analysis (PCA) to reduce the dimensionality of the data. These findings underscore the effectiveness of CNN models in audio classification tasks and highlight the limitations of Gaussian Naive Bayes in this context. Our work contributes to the growing field of audio classification by providing insights into the performance of various machine-learning models on an audio dataset.

Keywords: Audio Classification, TinySOL Dataset, Machine Learning Models, VGGish, K-Nearest-Neighbors (KNN), Support Vector Machine (SVM), Logistic Regression, Random Forest, Decision Tree, Gaussian Naive Bayes, Conv1D, Conv2D, and the Long Short-Term Memory (LSTM), CNN, PCA, Feature Dimensionality Reduction.

2. INTRODUCTION

The ability to automatically identify and categorize audio signals has become increasingly vital in our technologically driven world. From enhancing the efficiency of speech recognition systems and music recommendation platforms to improving the accuracy of security surveillance and environmental sound monitoring, machine learning-based audio classification stands at the forefront of solving critical challenges. The profound impact of audio classification transcends entertainment. It extends to domains such as healthcare, public safety, and industrial automation, where precise and efficient audio data handling plays a pivotal role. Machine learning has emerged as a transformative tool in audio classification, offering remarkable capabilities in deciphering complex audio data and contributing to many real-world applications.

3. MOTIVATION

In the ever-expanding landscape of digital media and the ubiquity of intelligent voice assistants, the realm of audio signal classification has taken on a pivotal role, offering an array of applications that range from transforming how we interact with our digital libraries to revolutionizing the healthcare industry. This report seeks to embark on a comprehensive exploration of machine learning techniques for audio signal classification, aiming to provide a panoramic view of the field's intricacies, recent advancements, persistent challenges, and a promising path forward. In light of these diverse applications and the potential to unlock even more, our objective is to equip readers with an understanding of the significance of audio classification in the modern world and the role that machine learning and deep learning play in realizing their full potential.

3.1 Growing Digital Media: With the exponential growth in digital media, there's a dire need to classify and catalog audio for efficient search, recommendation, and consumption. For instance, classifying music into genres or moods can help streaming services provide better user recommendations.

3.2 Intelligent Voice Assistants: As voice assistants like Siri, Alexa, and Google Assistant become more prevalent, the importance of accurate audio classification grows. Determining whether an input is a command, a question, or just background noise can dramatically improve the responsiveness and accuracy of these devices.

3.3 Healthcare Applications: Beyond the realm of entertainment and digital convenience, audio classification holds great promise in healthcare applications. The acoustic patterns of coughs, breaths, and vocal cords can serve as diagnostic clues for a wide range of medical conditions. From psychiatric ailments to respiratory diseases, the potential for audio classification to support remote diagnostics and telemedicine is immense.

4. BACKGROUND

In recent years, the field of audio classification has undergone a transformative shift, propelled by the surge in digital audio data and the application of sophisticated machine learning techniques. Fundamental to this domain is the representation of audio data, which often involves the extraction of pertinent features from raw signals. Time-domain representations, such as waveforms, offer insights into the temporal characteristics of sound, capturing nuances like rhythm and amplitude variations. Frequency-domain representations, on the other hand, provide a detailed analysis of the frequency components within an audio signal, making spectrograms a popular choice. Mel-frequency cepstral coefficients (MFCCs), derived from the spectral envelope of an audio signal, are particularly effective in capturing the human auditory system's sensitivity to different frequencies. Their ability to distill complex audio information into a compact feature set has led to widespread adoption, especially in speech and music classification tasks.

The advent of deep learning architectures has brought about a paradigm shift in audio classification. Convolutional Neural Networks (CNNs) excel in capturing hierarchical features in image-like representations, making them adept at processing spectrogram data. This has proven invaluable in tasks such as music genre classification, where patterns in the frequency domain are critical. Recurrent Neural Networks (RNNs), and in particular, Long Short-Term Memory (LSTM) networks, are instrumental in handling sequential dependencies in audio data. This is crucial in applications like speech recognition, where the temporal structure of spoken words is essential for accurate classification. Hybrid models, often combining CNNs and RNNs, leverage the strengths of both architectures to address the diverse and intricate nature of audio data, highlighting the

versatility of modern machine-learning techniques in tackling complex audio classification challenges.

5. METHOD

Our study examines ten classification models: VGGish, KNN, SVM, Logistic Regression, Random Forest, Decision Tree, GaussianNB, Conv1D, Conv2D, and LSTM. These models have been chosen for their potential in audio and music classification tasks, offering a diverse range of approaches for comparison. Prior research in the field has laid a foundational understanding of machine learning applications in audio classification. Our work aims to extend this knowledge by conducting a comparative analysis of these models, specifically in the context of the TinySOL dataset, to determine their efficacy in classifying instrument types and instrument families [1].

5.1 Dataset Overview

The TinySOL dataset is a collection of 2913 audio samples, each representing a single musical note played on one of 14 different instruments. Each audio file is stored as a WAV file, sampled at 44.1 kHz, with a single channel (mono), a bit depth of 16, and a duration that varies between 2 and 10 seconds. The dataset is organized into instrument families (Brass, Keyboards, Strings, and Winds), individual instruments (Bass Tuba, French Horn, Trombone, Trumpet in C, Accordion, Contrabass, Violi, Viola, Violoncello, Bassoon, Clarinet in B-flat, Flute, Oboe, and Alto Saxophone), and pitch [1].

5.2 Baseline Model – Support Vector Machines (SVM)

Audio data, when transformed into feature vectors, can be high-dimensional. SVMs are well-suited for high-dimensional data, especially using the Radial Basis Function (RBF) kernel [2]. While inherently binary, SVMs can be extended to handle multi-class classification, making them versatile for tasks that involve multiple audio classes. Given the reasons above, SVM was chosen as our baseline model.

5.3 Additional Models Trained & Evaluated

K-Nearest Neighbors (KNN): K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm. This model was used to classify audio samples based on their similarity to neighboring samples. In audio classification, KNN operates by finding the `k` nearest audio samples to a given sample and assigning it to the most common class among its K-Nearest Neighbors. The parameter `k` represents the number of neighbors to consider when making a classification decision. In our preliminary tests, the value of `k` was set to 5. KNN is particularly useful when the dataset exhibits local patterns, where samples of the same class tend to be close to each other in the feature space. The performance of the KNN algorithm can vary significantly depending on the value of K selected. Smaller values of `k` (e.g., `k=1` or `k=3`) make the model more sensitive to local variations in the data, which can lead to noisy less stable predictions. It might overfit the training data because it relies heavily on the nearest neighbors. Larger values of `k` (e.g., `k=20` or `k=50`) make the model less sensitive to local fluctuations and more influenced by the global structure of the data. Larger values of `k` can lead to smoother decision boundaries but might cause underfitting, especially when there are complex patterns in the data.

Logistic Regression: Logistic Regression is a linear model widely used for binary classification tasks, but it can also be extended to multiclass classification, making it suitable for our audio classification problem with 14 instrument classes. Logistic Regression models the relationship between audio features and the likelihood of an audio sample belonging to a particular class. It applies the logistic (sigmoid) function to a linear combination of input features. While it's a simple model, it's known for its interpretability and is a useful benchmark for multiclass classification.

In Logistic Regression, the 'solver' used was `lbfgs` (Limited-memory Broyden-Fletcher-Goldfarb-Shanno). `Lbfgs` is a solver that approximates the second derivative matrix updates using only a small number of gradient vectors. Also, the maximum iterations parameter was set to 1000. When increasing the parameter to 10,000 iterations, the model took more than 20 minutes to train and still failed to converge.

Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to create a more powerful model. It is capable of handling both classification and regression tasks. In our audio classification scenario, Random Forest captured complex relationships within the audio data. It operates by constructing a forest of decision trees, each tree contributing to the final classification. Random Forest mitigates the overfitting issues often seen with individual decision trees.

A primary parameter for a Random Forest model is the number of decision trees in the forest. In `scikit-learn`, this is denoted by the parameter `n_estimators`. In our case, 100 decision trees are used. We anticipate experimenting with different values of `n_estimators` to determine the optimal number of trees for the TinySOL dataset. More trees can lead to better generalization but may require more computational resources. Other parameters might be included in future work, such as `max_depth` and `max_samples` (these were kept at the default `scikit-learn` values for the initial experiments outlined in this report).

Decision Tree Classifier: The Decision Tree Classifier is a model that recursively splits the audio dataset into subsets based on the most discriminative audio features. Each internal tree node represents a feature, and each leaf node represents a class label. Decision trees effectively capture local patterns in the data, making them suitable for audio classification. However, they can be prone to overfitting, and the depth of the tree should be carefully controlled. The default settings of the `sklearn` were used. Fine-tuning might involve adjusting parameters like `max_depth`, `min_samples_split`, or `min_samples_leaf`, (but doing so led to a decrease in performance in our case, so the default parameters were reported on).

Gaussian Naive Bayes: Gaussian Naive Bayes is a probabilistic model that works well for multiclass classification tasks, including audio classification. It assumes that the audio features are

normally distributed and independent within each class. By estimating the parameters of these Gaussian distributions, the model can calculate the posterior probability of an audio sample belonging to a specific class. Gaussian Naive Bayes follows the assumption that features follow a Gaussian distribution. While it doesn't have explicit parameters like `k` in KNN or `C` in SVM, one can preprocess the data and reduce its dimensionality to better align with the Gaussian distribution assumption and thus improve performance (potentially).

VGGish Model: The VGGish model, adapted from the influential VGG architecture originally developed for image recognition, is a convolutional neural network specifically tailored for audio processing. It transforms audio input, typically 0.96-second log-mel spectrograms, into high-level features through a series of convolutional layers. Primarily used for feature extraction, VGGish is instrumental in various audio analysis applications, including sound classification, audio event detection, and music analysis. Its strength lies in its ability to handle complex audio data, and the availability of pre-trained models facilitates its use in transfer learning scenarios. We leveraged transfer learning to retrain the VGGish model, a pre-trained convolutional neural network available in MatLab's Audio Toolbox, to classify the new set of audio signals from the TinySOL dataset.

Conv1D Model: The Conv1D model represents a sophisticated audio-processing neural network, constructed using the Keras API. The architecture initiates with a mel-spectrogram layer to transform audio inputs into a time-frequency domain, configuring 128 mel bands and optimizing windowing parameters (n_fft=512, win_length=400, hop_length=160) for spectral decomposition. This layer is pivotal in extracting relevant features from raw audio data. Subsequent layers consist of a series of TimeDistributed Conv1D layers, escalating in complexity (8 to 128 filters), interspersed with MaxPooling2D layers for feature abstraction and dimensionality reduction. These layers are integral in learning temporal characteristics from the transformed audio signal.

The network employs GlobalMaxPooling2D for spatial feature reduction, followed by a Dropout layer (rate=0.1) to mitigate overfitting. The penultimate layer is a dense layer with L2 regularization, facilitating feature integration before classification. The final layer, a softmax activation-based dense layer, is designed for multi-class categorization, correlating to `N_CLASSES`. Compiled with an Adam optimizer and categorical cross-entropy loss, the model's architecture is apt for intricate audio signal classification tasks, balancing between feature extraction efficiency and computational feasibility.

Conv2D Model: The Conv2D model is a specialized convolutional neural network devised for advanced audio signal processing, employing a 2D convolutional approach. Following the spectrogram layer used in the Conv1D model, this model employs a series of 2D convolutional layers (Conv2D), beginning with an 8-filter layer using a tanh activation function, and progressively increasing in complexity and depth, with subsequent layers utilizing ReLU activation functions. These layers are interspersed with MaxPooling2D layers, which aid in reducing dimensionality and extracting salient features from the spectrogram. Then further refined with a Flatten layer, Dropout layer, and softmax layer same as Conv1D.

Model: "1d_convolution"

Layer (type)	Output Shape	Param #
stft_2_input (InputLayer)	[(None, 16000, 1)]	0
stft_2 (STFT)	(None, 100, 257, 1)	0
magnitude_2 (Magnitude)	(None, 100, 257, 1)	0
apply_filterbank_2 (ApplyFilterbank)	(None, 100, 128, 1)	0
magnitude_to_decibel_2 (MagnitudeToDecibel)	(None, 100, 128, 1)	0
batch_norm (LayerNormalization)	(None, 100, 128, 1)	256
td_conv_1d_tanh (TimeDistributed)	(None, 100, 125, 8)	40
max_pool_2d_1 (MaxPooling2D)	(None, 50, 62, 8)	0
td_conv_1d_relu_1 (TimeDistributed)	(None, 50, 59, 16)	528
max_pool_2d_2 (MaxPooling2D)	(None, 25, 29, 16)	0
td_conv_1d_relu_2 (TimeDistributed)	(None, 25, 26, 32)	2080
max_pool_2d_3 (MaxPooling2D)	(None, 12, 13, 32)	0
td_conv_1d_relu_3 (TimeDistributed)	(None, 12, 10, 64)	8256
max_pool_2d_4 (MaxPooling2D)	(None, 6, 5, 64)	0
td_conv_1d_relu_4 (TimeDistributed)	(None, 6, 2, 128)	32896
global_max_pooling_2d (GlobalMaxPooling2D)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8256
softmax (Dense)	(None, 10)	650

Model: "2d_convolution"

Layer (type)	Output Shape	Param #
stft_3_input (InputLayer)	[(None, 16000, 1)]	0
stft_3 (STFT)	(None, 100, 257, 1)	0
magnitude_3 (Magnitude)	(None, 100, 257, 1)	0
apply_filterbank_3 (ApplyFilterbank)	(None, 100, 128, 1)	0
magnitude_to_decibel_3 (MagnitudeToDecibel)	(None, 100, 128, 1)	0
batch_norm (LayerNormalization)	(None, 100, 128, 1)	256
conv2d_tanh (Conv2D)	(None, 100, 128, 8)	400
max_pool_2d_1 (MaxPooling2D)	(None, 50, 64, 8)	0
conv2d_relu_1 (Conv2D)	(None, 50, 64, 16)	3216
max_pool_2d_2 (MaxPooling2D)	(None, 25, 32, 16)	0
conv2d_relu_2 (Conv2D)	(None, 25, 32, 16)	2320
max_pool_2d_3 (MaxPooling2D)	(None, 13, 16, 16)	0
conv2d_relu_3 (Conv2D)	(None, 13, 16, 32)	4640
max_pool_2d_4 (MaxPooling2D)	(None, 7, 8, 32)	0
conv2d_relu_4 (Conv2D)	(None, 7, 8, 32)	9248
flatten (Flatten)	(None, 1792)	0
dropout (Dropout)	(None, 1792)	0
dense (Dense)	(None, 64)	114752
softmax (Dense)	(None, 10)	650

Figure 1: CNN Layer Architecture of Model Conv1D and Conv2D

LSTM Model: The LSTM model delineates an intricate neural network architecture specifically designed for processing audio data, utilizing Long Short-Term Memory (LSTM) layers within the Keras framework. This model stands out for its ability to capture temporal dependencies in audio signals, a critical aspect for tasks involving time-series data like sound classification. A significant feature of this model is the incorporation of a bidirectional LSTM layer, allowing the model to learn from the audio data in both forward and backward directions, thereby capturing a broader context. This layer is preceded by a time-distributed dense layer (64 units, tanh activation) to provide a rich feature set to the LSTM layer. A skip connection is then employed, concatenating the output of the initial time-distributed layer with the LSTM output, enhancing the model's ability to preserve information from earlier layers.

Model: "long_short_term_memory"

Layer (type)	Output Shape	Param #	Connected to
stft_4_input (InputLayer)	[(None, 16000, 1)]	0	[]
stft_4 (STFT)	(None, 100, 257, 1)	0	['stft_4_input[0][0]']
magnitude_4 (Magnitude)	(None, 100, 257, 1)	0	['stft_4[0][0]']
apply_filterbank_4 (ApplyFilterbank)	(None, 100, 128, 1)	0	['magnitude_4[0][0]']
magnitude_to_decibel_4 (MagnitudeToDecibel)	(None, 100, 128, 1)	0	['apply_filterbank_4[0][0]']
batch_norm (LayerNormalization)	(None, 100, 128, 1)	256	['magnitude_to_decibel_4[0][0]']
reshape (TimeDistributed)	(None, 100, 128)	0	['batch_norm[0][0]']
td_dense_tanh (TimeDistributed)	(None, 100, 64)	8256	['reshape[0][0]']
bidirectional_lstm (Bidirectional)	(None, 100, 64)	24832	['td_dense_tanh[0][0]']
skip_connection (Concatenate)	(None, 100, 128)	0	['td_dense_tanh[0][0]', 'bidirectional_lstm[0][0]']
dense_1_relu (Dense)	(None, 100, 64)	8256	['skip_connection[0][0]']
max_pool_1d (MaxPooling1D)	(None, 50, 64)	0	['dense_1_relu[0][0]']
dense_2_relu (Dense)	(None, 50, 32)	2080	['max_pool_1d[0][0]']
flatten (Flatten)	(None, 1600)	0	['dense_2_relu[0][0]']
dropout (Dropout)	(None, 1600)	0	['flatten[0][0]']
dense_3_relu (Dense)	(None, 32)	51232	['dropout[0][0]']
softmax (Dense)	(None, 10)	330	['dense_3_relu[0][0]']

Figure 2: CNN Layer Architecture of Model LSTM

5.4 Metrics: Evaluating Model Performance: The metrics outlined below were used to evaluate the performance of the models on the validation set.

Accuracy: Accuracy measures the overall correctness of the model's predictions. It's the ratio of the number of correctly predicted instances to the total number of instances. A higher accuracy indicates better overall performance.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Precision: Precision measures the accuracy of positive predictions made by the model. It calculates the ratio of true positives to the sum of true positives and false positives. Precision tells how many of the positive predictions made by the model were correct. It's especially important when false positives are costly or when it's important to ensure that the positive predictions are highly accurate.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

F1-Score: The F1-Score considers both false positives and false negatives, providing a single metric that considers both precision and recall. It provides a balanced measure of the model's performance by considering false positives and negatives.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F1\ Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Confusion Matrix: A confusion matrix is a table used to describe the performance of a classification algorithm. It consists of four values: *true positives* (the number of correctly predicted positive instances), *true negatives* (the number of correctly predicted negative instances), *false positives* (the number of negative instances incorrectly classified as positive, a Type I error), and *false negatives* (the number of positive instances incorrectly classified as negative, a Type II error).

In scenarios involving imbalanced datasets such as the TinySOL dataset, selecting the appropriate evaluation metric becomes crucial for a comprehensive assessment of model performance. In this context, we focus on metrics that better address the challenges of imbalanced class sample sizes in the dataset. Particularly, the F1 Score considers the trade-off between false positives and false negatives, offering a balanced performance indicator. Additionally, we rely on the confusion matrix to provide a granular breakdown of true positives, true negatives, false positives, and false negatives for each class, ensuring a detailed analysis of the model's classification capabilities.

5.5 Model Training and Validation

In this study, we employed a consistent approach for training and validating all ten classification models: VGGish, KNN, SVM, Logistic Regression, Random Forest, Decision Tree, GaussianNB, Conv1D, Conv2D, and LSTM. To ensure uniformity and comparability across models, the TinySOL dataset was divided using an 8:2 split ratio. Specifically, 80% of the data was allocated for model training, while the remaining 20% was reserved for validation and testing purposes.

6. EXPERIMENTAL SETUP

6.1 Data Preprocessing

Before preprocessing the TinySOL audio dataset, several critical data processing steps were taken to prepare the audio data for input into machine learning models. Given the different nature and data input requirements (CNN models require at least a 3-dimensional input matrix) for different models, different data processing techniques were employed for CNN models and other models.

6.1.1 VGGish, KNN, SVM, Logistic Regression, Random Forest, Decision Tree, GaussianNB models:

Audio File Loading: Initially, the audio data from the TinySOL dataset was loaded and converted into a numerical format suitable for machine learning.

Feature Extraction: Relevant audio features were extracted, including Mel-frequency cepstral coefficients (MFCCs) and chroma features. MFCCs capture spectral characteristics, while chroma features represent the pitch class of musical notes. The `Librosa` library in Python was used to extract Mel-frequency cepstral coefficients (MFCCs) from each segment. MFCCs are commonly used audio features for speech and audio processing tasks. The extracted MFCCs are transposed to have a shape of `(time_steps, num_features)`. This shape is suitable for input into machine learning models.

Normalization: The extracted features were normalized to ensure consistent scales, facilitating the training of machine learning models.

Label Encoding: The labels for the 14 musical instruments were encoded numerically, a prerequisite for many machine learning algorithms.

Data Flattening: To handle audio files of varying lengths, the extracted features were flattened into fixed-size feature vectors. This step guaranteed that all samples had consistent dimensions for input into the machine learning models.

Data Splitting and Balancing: After feature extraction, the dataset was split into training and validation subsets with an 80-20% division, respectively. Special care was taken to balance the class distribution in both sets using the `StratifiedShuffleSplit` method from `scikit-learn`. This balance helped reduce model bias towards majority classes and enabled better generalization.

Class	Training Set	Validation Set
Accordion	0.243506	0.254814
Alto Saxophone	0.037841	0.037170
Bass Tuba	0.015674	0.018361
Bassoon	0.034819	0.038065
Cello	0.112405	0.112853
Clarinet in Bb	0.046350	0.040752
Contrabass	0.092924	0.095835
Flute	0.043103	0.033587
French Horn	0.045567	0.047918
Oboe	0.038401	0.036722
Trombone	0.031908	0.029557
Trumpet in C	0.032132	0.034931
Viola	0.114308	0.115092
Violin	0.111061	0.104344

Figure 3. The ratios of each class in the training and validation sets

Data Shuffling: Random shuffling of samples in both training and validation sets helps mitigate any biases arising from the original order of the dataset.

Audio Segmentation and Padding: Special attention was given to ensure that all audio samples had a uniform duration of 2 seconds. Audio segmentation involved dividing each audio sample into smaller segments, each 2 seconds in duration. For audio files that were already 2 seconds, these files retained their original duration. For audio files longer than 2 seconds, they were segmented into non-overlapping 2-second segments. Silent padding was employed for segments shorter than 2 seconds, adding silent audio to achieve the uniform 2-second duration.

6.1.2 Steps Specific to Conv1D, Conv2D, and LSTM models only:

Audio File Loading: Audio File Loading: The process of loading audio data involved reading from the specified file path using the function **`downsample_mono(path, sr)`**. This function accepts the file path and sampling rate as input parameters and performs the conversion of the audio into a single channel (mono) in case it is originally in stereo format. To ensure consistency and simplify subsequent processing, the sampling rate was uniformly set to 1 kHz. This was achieved through the utilization of the built-in function **`librosa.resample(wav, orig_sr=rate, target_sr=sr)`**, which takes the audio waveform '**`wav`**,' original sampling rate '**`rate`**,' and the target sampling rate '**`sr`**' as parameters, returning the resampled audio along with the updated file path and sampling rate. The decision to load only mono-channel samples is deliberate and aligns with the pragmatic goal of simplifying the data for improved model manageability. By restricting the audio data to a single channel, we not only ensure uniformity in the format but also reduce the complexity of the input, facilitating more efficient processing by our machine learning models. This choice streamlines subsequent feature extraction and model training, promoting a more straightforward and effective audio analysis pipeline.

Audio Signal Threshold: The Audio Signal Threshold function, denoted as **`envelope(y, rate, threshold)`**, serves the purpose of identifying the presence of audio signals surpassing a specified threshold within an audio signal '**`y`**' sampled at a given rate. This threshold effectively demarcates where the audio signal concludes. When the peaks of the audio signal are below the threshold, the data is deemed ready for further processing. Conversely, if the peaks lag above the threshold, it necessitates adjusting the threshold to ensure the inclusion of all pertinent signals. Our initial threshold was set at 100, but through iterative refinement, we ultimately determined an optimal threshold of 20. The adjustment of the threshold assumes paramount importance as it plays a pivotal role in optimizing the workload by selectively retaining essential data. A judiciously chosen threshold ensures that the processing focuses on meaningful audio signals, thereby enhancing efficiency. This adaptability in threshold adjustment is crucial in scenarios where the temporal dynamics of audio signals may vary, allowing for a dynamic and responsive system that retains essential information while minimizing extraneous processing.

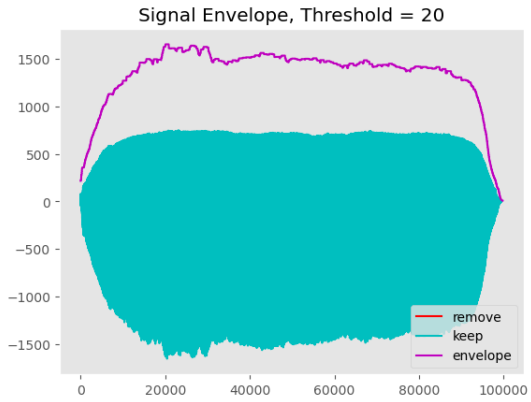

Figure 4. Singal Envelope with a threshold set at 20

Sample Saving: The function **`save_sample(sample, rate, target_dir, fn, ix)`** is designed to store a segment of an audio file (**`sample`**) at a specified sampling rate (**`rate=1.0`**) within the designated target directory (**`target_dir`**). The filename is modified based on the original file names (**`fn`**) and a stringified index (**`ix`**). This approach ensures that data can be consistently computed at the same file location during iterative processes, allowing for seamless computation without the need to alter file paths.

Splitting Wavs: the function **`split_wavs(args)`** is the main function that executes the data cleaning and preparation process. By calling this function, we go through each step above to iterate through every class in the source directory, then process and save the audio samples.

Label Encoding: The labels for the 14 musical instruments were encoded numerically, just as explained previously for the other models.

Data Reshaping: Following the encoding of data labels, the subsequent step involves the generation of a batch of time data, denoted by parameters X (representing the time values of the batch) and Y (indicating the corresponding labels for each batch). The pivotal transformation occurs as we reshape each WAV file

into a 3-dimensional input matrix, incorporating the X and Y parameters.

To provide technical depth, this process involves structuring the data in a format suitable for input into machine learning models. The creation of a time-based batch allows us to organize the temporal aspects of the audio data effectively. Reshaping each WAV file into a 3-dimensional input matrix implies converting the raw audio data into a format compatible with the model's input requirements. This 3D matrix typically encompasses dimensions such as time, frequency, and channels. The precise details of reshaping depend on the specific model architecture and input expectations. Overall, this reshaping operation is pivotal in preparing the data for subsequent processing and training within the machine learning framework.

7. OUTCOMES & RESULTS

7.1 KNN, SVM, Logistic Regression, Random Forest, Decision Tree, and Gaussian Naive Bayes models

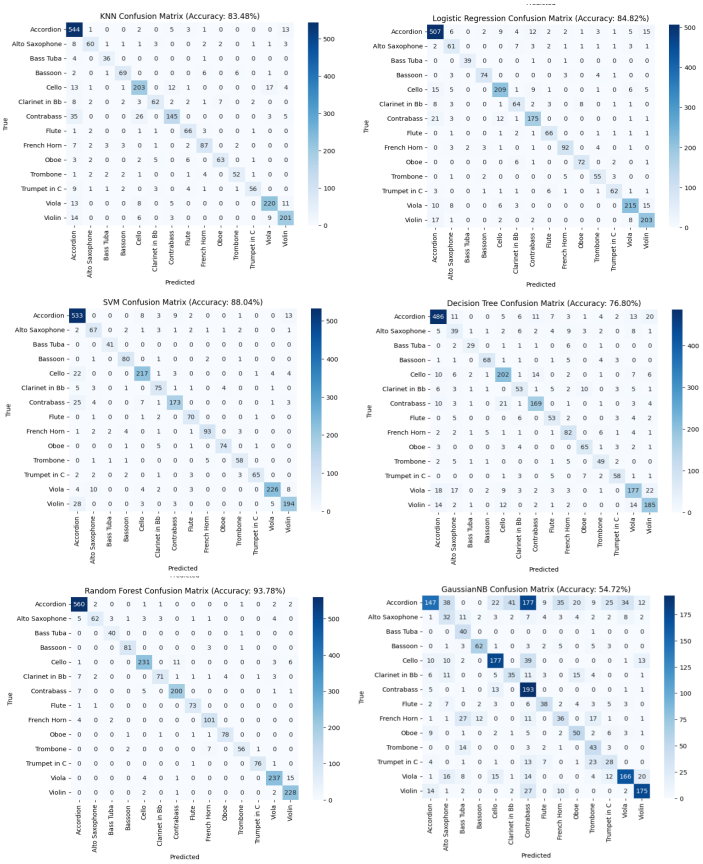

Figure 5. Confusion matrix of KNN, SVM, Logistic Regression, Random Forest, Decision Tree, and Gaussian Naive Bayes

To provide more granularity, we broke down the confusion matrix of each model, which revealed the number of correct and incorrect predictions per class, aiding in pinpointing areas of strength and potential weakness for each model.

Confusion Matrix Summaries:

Model: KNN

Class	Correct	Incorrect	Samples
Accordion	544	25	569
Alto Saxophone	60	23	83
Bass Tuba	36	5	41
Bassoon	69	16	85
Cello	203	49	252
Clarinet in Bb	62	29	91
Contrabass	145	69	214
Flute	66	9	75
French Horn	87	20	107
Oboe	63	19	82
Trombone	52	14	66
Trumpet in C	56	22	78
Viola	220	37	257
Violin	201	32	233

Model: Random Forest

Class	Correct	Incorrect	Samples
Accordion	562	7	569
Alto Saxophone	66	17	83
Bass Tuba	40	1	41
Bassoon	82	3	85
Cello	232	20	252
Clarinet in Bb	72	19	91
Contrabass	202	12	214
Flute	73	2	75
French Horn	101	6	107
Oboe	78	4	82
Trombone	56	10	66
Trumpet in C	76	2	78
Viola	237	20	257
Violin	225	8	233

Model: SVM

Class	Correct	Incorrect	Samples
Accordion	533	36	569
Alto Saxophone	67	16	83
Bass Tuba	41	0	41
Bassoon	80	5	85
Cello	217	35	252
Clarinet in Bb	75	16	91
Contrabass	173	41	214
Flute	70	5	75
French Horn	93	14	107
Oboe	74	8	82
Trombone	58	8	66
Trumpet in C	65	13	78
Viola	226	31	257
Violin	194	39	233

Model: Decision Tree

Class	Correct	Incorrect	Samples
Accordion	484	85	569
Alto Saxophone	35	48	83
Bass Tuba	29	12	41
Bassoon	66	19	85
Cello	202	50	252
Clarinet in Bb	55	36	91
Contrabass	175	39	214
Flute	58	17	75
French Horn	82	25	107
Oboe	65	17	82
Trombone	47	19	66
Trumpet in C	58	20	78
Viola	182	75	257
Violin	187	46	233

Model: Logistic Regression

Class	Correct	Incorrect	Samples
Accordion	507	62	569
Alto Saxophone	61	22	83
Bass Tuba	39	2	41
Bassoon	74	11	85
Cello	209	43	252
Clarinet in Bb	64	27	91
Contrabass	175	39	214
Flute	66	9	75
French Horn	92	15	107
Oboe	72	10	82
Trombone	55	11	66
Trumpet in C	62	16	78
Viola	215	42	257
Violin	203	30	233

Model: GaussianNB

Class	Correct	Incorrect	Samples
Accordion	147	422	569
Alto Saxophone	32	51	83
Bass Tuba	40	1	41
Bassoon	62	23	85
Cello	177	75	252
Clarinet in Bb	35	56	91
Contrabass	193	21	214
Flute	38	37	75
French Horn	36	71	107
Oboe	50	32	82
Trombone	43	23	66
Trumpet in C	28	50	78
Viola	166	91	257
Violin	175	58	233

Figure 6. Confusion Matrix Summaries for Model Performances on Validation Set

	Model	Accordion	Alto Saxophone	Bass Tuba	Bassoon	Cello	Clarinet in Bb	Contrabass	Flute	French Horn	Oboe	Trombone	Trumpet in C	Viola	Violin
Correct %	KNN	95.61	72.29	87.80	81.18	80.56	68.13	67.76	88.00	81.31	76.83	78.79	71.79	85.60	86.27
Error %	KNN	4.39	27.71	12.20	18.82	19.44	31.87	32.24	12.00	18.69	23.17	21.21	28.21	14.40	13.73
Correct %	SVM	93.32	80.72	97.56	95.29	86.90	80.22	78.97	93.33	87.85	91.46	87.88	83.33	84.05	86.27
Error %	SVM	6.68	19.28	2.44	4.71	13.10	19.78	21.03	6.67	12.15	8.54	12.12	16.67	15.95	13.73
Correct %	Logistic Regression	88.40	74.70	95.12	88.24	84.13	71.43	83.18	89.33	86.92	87.80	84.85	80.77	84.82	87.55
Error %	Logistic Regression	11.80	25.30	4.88	11.76	15.87	28.57	16.82	10.67	13.08	12.20	15.15	19.23	15.18	12.45
Correct %	Random Forest	98.77	78.31	97.56	97.65	92.86	82.42	93.46	97.33	91.59	96.34	87.88	97.44	92.61	96.57
Error %	Random Forest	1.23	21.69	2.44	2.35	7.14	17.58	6.54	2.67	8.41	3.66	12.12	2.56	7.39	3.43
Correct %	Decision Tree	84.53	43.37	68.29	76.47	79.76	58.24	79.91	62.67	76.64	75.61	80.30	74.36	68.93	78.11
Error %	Decision Tree	15.47	56.63	31.71	23.53	20.24	41.76	20.09	37.33	23.36	24.39	19.70	25.64	33.07	21.89
Correct %	GaussianNB	25.83	38.55	97.56	72.94	70.24	38.46	90.19	50.67	33.64	60.98	65.15	35.90	64.59	75.11
Error %	GaussianNB	74.17	61.45	2.44	27.06	29.76	61.54	9.81	49.33	66.36	39.02	34.85	64.10	35.41	24.89

Figure 7. Percentage Table of Correct and Error Rates

Model Performance Metrics:

	Model	Accuracy	Precision	F1 Score
3	Random Forest	0.9413345275414241	0.941427643084649	0.9406683892237759
1	SVM	0.8804299149126735	0.8833115658499077	0.8804480213124297
2	Logistic Regression	0.8481862964621585	0.8497282867750892	0.8485027351873957
0	KNN	0.8347514554411106	0.8364854275288304	0.8323460353378793
4	Decision Tree	0.7725033587102552	0.7779685553982592	0.7741144935956044
5	GaussianNB	0.5472458575906852	0.6193715368354756	0.5389950476715758

Figure 8. Model Performance Metrics

The Performance of these models varies differently from an accuracy of 0.9413 to 0.5472, precision varies from 0.9414 to 0.6194, and F1 Score varies from 0.9407 to 0.5390. Random Forest behaved the best while GaussianNB had the worst performance. SVM, Logistic Regression, and KNN had a similar performance of accuracy about 0.85, and Decision Tree had a slightly lower accuracy than KNN, which was 0.7725.

7.2 The VGGish Model

Following a preliminary assessment of the above models, we began exploring more advanced approaches tailored to the task of audio classification. We opted for the VGGish model due to its specialized architecture designed for audio data processing. Derived from the renowned VGG network, VGGish stands out in its ability to extract intricate audio features through its deep convolutional layers. To utilize the VGGish model, we leveraged MATLAB's Audio Toolbox which includes a pre-trained VGGish model. We prepared the network for transfer learning by replacing its final layers with new ones tailored to our dataset.

After training and testing the network on our dataset, we achieved an accuracy rate of 80.446%.

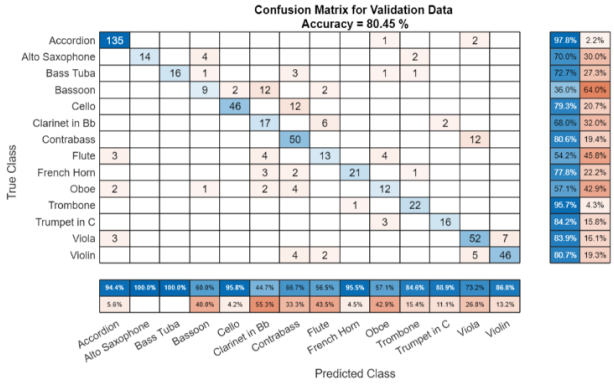

Figure 9. VGGish Confusion Matrix

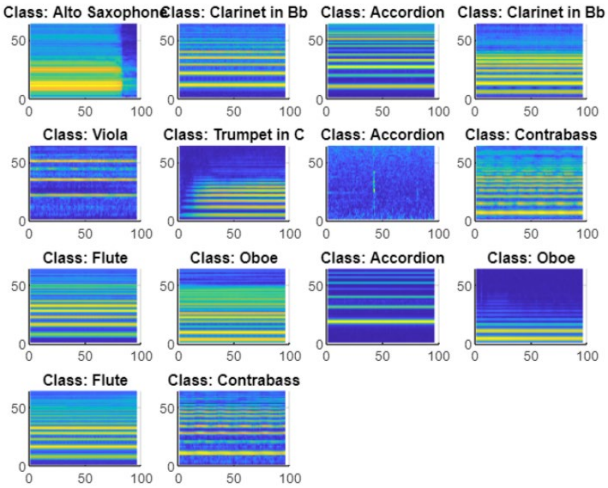

Figure 10: The VGGish pre-trained network requires the audio signals to be transformed into log mel spectrograms. The time is depicted on the x-axis, the frequency on the y-axis, and the colormap signifies the decibel values. In various classes, discernible features are observable.

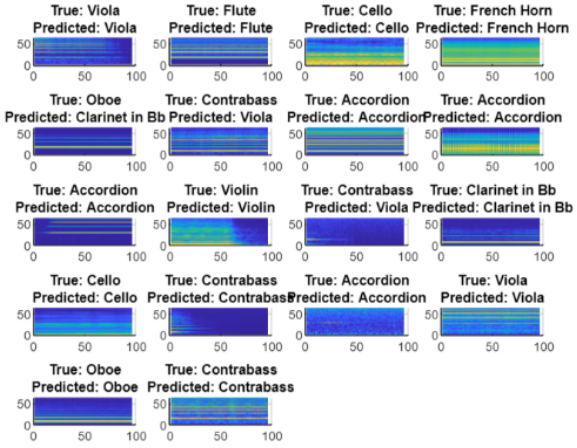

Figure 11: VGGish Prediction Visualization on a Sample Subset of the Validation Set

7.3 Conv1D, Conv2D, and LSTM Models

After an initial exploration of VGGish, subsequent experimentation delved into additional Convolutional Neural Network (CNN) models for the analysis of the audio dataset and the refinement of model predictions. In this iterative process, Conv1D, Conv2D, and LSTM (Long Short-Term Memory), a recurrent neural network architecture widely used in Deep Learning, were implemented. Impressively, each of these models exhibited remarkable accuracy, consistently surpassing a performance level of 97%.

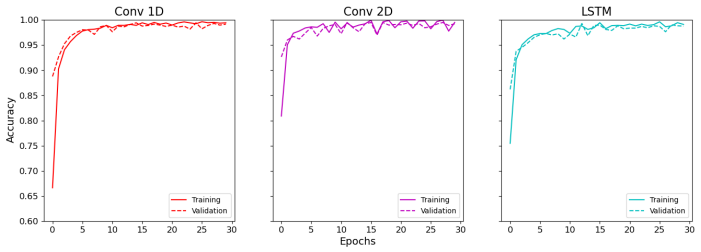

Figure 12: Training and Validation Accuracy of Model Conv1D, Conv2D, and LSTM

As illustrated in the accompanying graph and table, Conv2D emerges as the frontrunner, boasting the highest accuracy and mean accuracy, indicative of superior overall performance. While both LSTM and Conv1D demonstrate marginally lower accuracy levels compared to Conv2D, they outshine all other models investigated in the study. Particularly noteworthy is the fact that LSTM and Conv1D consistently achieve accuracy levels ranging between 97% and 98%, accompanied by heightened F1 scores and precision rates that extend up to 98%. This robust performance underscores the effectiveness of these CNN and LSTM architectures in the context of audio classification.

Model	Conv1D	Conv2D	LSTM
Best Val Accuracy	0.994079	0.994737	0.992763
Corresponding Val Loss	0.042673	0.049231	0.058892
Mean Val Accuracy	0.979232	0.982171	0.972478
Median Val Accuracy	0.987171	0.988487	0.979934
Epoch	30	30	30
Best Epoch at	14.000000	11.000000	12.000000
F1 Score	0.982310	0.997381	0.987211
Precision	0.991231	0.997214	0.973137

Figure 13: Model Performance Table with Accuracy, F1 Score, Precision, and Epoch

In this multi-class classification experiment utilizing audio data, the evaluation of machine-learning model performance is facilitated through the utilization of both the Receiver Operating Characteristic (ROC) curve and the confusion matrix (graph displayed on the next page).

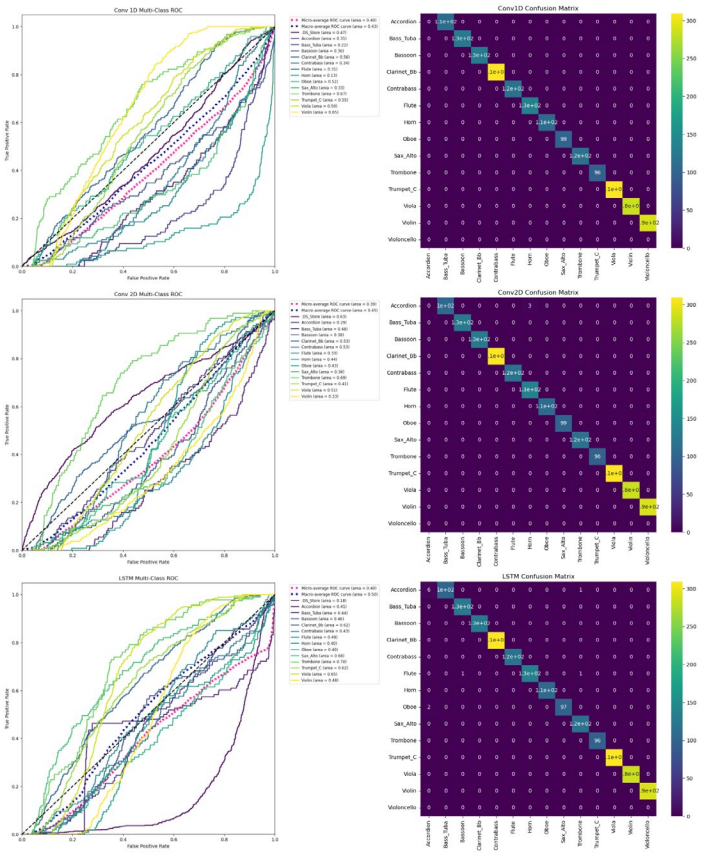

Figure 14: ROC curve and Confusion matrix of Model Conv1D, Conv2D, and LSTM

After the initial analysis, further performance investigations were conducted by examining the ROC curve and the confusion matrix. Notably, the diagonal of the confusion matrix exhibits exceptionally high values, while non-diagonal elements predominantly register values of 0 or 1. This pattern elucidates the accuracy and commendable performance of the models. However, deeper insights gleaned from ROC curves reveal lower micro and macro-average Area Under the Curve (AUC) values, indicating potential challenges in model generalization across all classes. The disparate AUC values for individual classes suggest varying degrees of model distinguishability, potentially attributed to class imbalances or variations in feature complexity among different classes.

The observed variability in class-specific performance across all models might stem from several factors, including imbalanced datasets, inherent similarities between specific instrument sounds, and potential suboptimal configuration of the model architecture and hyperparameters for certain classes. This nuanced analysis underscores the importance of considering not only overall model accuracy but also class-specific performance metrics when interpreting the effectiveness of machine-learning models in audio classification tasks.

7.4 Model Fine-Tuning

For the KNN model, we implemented a loop function that iterates through `k` values from 1 to 20 to determine the optimal `k` value for the model.

k Value	Accuracy	Precision	Recall	F1 Score
1	0.862069	0.862277	0.862069	0.861366
2	0.852665	0.856052	0.852665	0.851068
3	0.842812	0.843917	0.842812	0.841097
4	0.836991	0.838923	0.836991	0.834646
5	0.834751	0.836529	0.834751	0.832343
6	0.829825	0.833911	0.829825	0.827125
7	0.825347	0.829007	0.825347	0.823062
8	0.821317	0.825017	0.821317	0.818820
9	0.816838	0.821452	0.816838	0.814298
10	0.813704	0.819813	0.813704	0.811315
11	0.810121	0.816940	0.810121	0.807687
12	0.803851	0.812121	0.803851	0.801571
13	0.801612	0.808986	0.801612	0.799168
14	0.798030	0.808336	0.798030	0.795537
15	0.791312	0.800975	0.791312	0.788794
16	0.788625	0.798560	0.788625	0.785399
17	0.783699	0.793722	0.783699	0.780689
18	0.777877	0.788352	0.777877	0.774939
19	0.774295	0.785097	0.774295	0.771172
20	0.765338	0.775461	0.765338	0.761674

Figure 15: KNN Model Performance Metrics for values of k 1-20

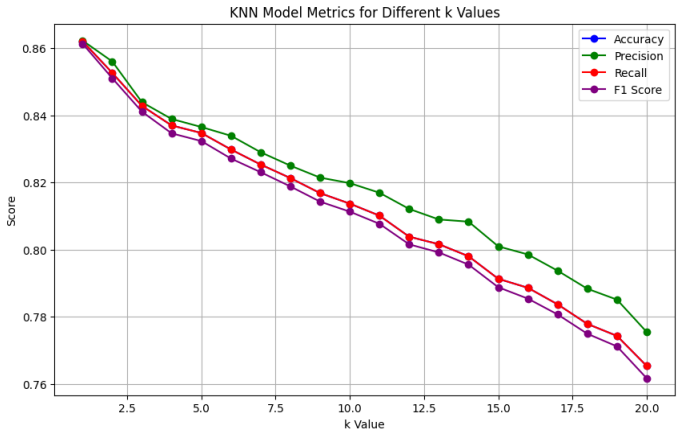

Figure 16: Trend of KNN Model Performance Metric for values of k 1 through 20

The optimal value of k was observed for `k = 1`, where accuracy, precision, and the f-1 score (0.862069, 0.862277, and 0.861366, respectively) were the highest when the model was used to predict the instrument on the test set. As the k-value increased, model performance decreased linearly as observed on the graph in Figure 15.

7.5 Reduce High Dimensionality of Features

The dimensionality of features plays a crucial role in influencing the performance of machine learning models, and its impact varies across different algorithms. To assess this impact, we applied Principal Component Analysis (PCA) for dimensionality reduction across the following models: Random Forest, SVM, Logistic Regression, KNN, Decision Trees, and Gaussian Naive Bayes. The primary goal was to decrease the number of features while retaining essential information. Notably, various models exhibit diverse sensitivities to high-dimensional features, with particular attention given to the Gaussian Naive Bayes model due to its known sensitivity to dimensionality.

Contrary to our expectations, implementing PCA resulted in a decrease in the accuracy of these six models, namely: Random Forest, SVM, Logistic Regression, KNN, Decision Trees, and notably, the Gaussian Naive Bayes model. Our specific focus on investigating the impact of PCA on the Gaussian Naive Bayes

model revealed a surprising drop in accuracy to approximately 0.40, compared to the original accuracy of 0.55 without PCA. Reflecting on these unexpected outcomes, we identified potential reasons for the observed failures.

Firstly, PCA operates by capturing the most significant variance in the data, yet this may not align with the most critical features for the predictive task at hand. In the context of our audio file dataset, the components removed during dimensionality reduction may contain valuable information vital for the models, resulting in information loss and compromised accuracy. PCA, in essence, seeks orthogonal linear combinations of features to capture maximum variance in the data. While it does assume linearity in this particular manner, it does not make explicit assumptions about the nature of relationships between the original features in terms of linearity or non-linearity. The potential inefficacy of PCA in capturing non-linear relationships arises due to its reliance on linear transformations. When the underlying relationships in the data are highly non-linear, PCA may struggle to accurately represent the data's structure with a limited number of principal components. In such cases, non-linear dimensionality reduction techniques like t-distributed Stochastic Neighbor Embedding (t-SNE) or Isomap might be more appropriate. Therefore, the consideration of linearity in PCA primarily pertains to its mechanism of capturing variance through linear combinations, and its performance may be suboptimal when the relationships between features in the data are inherently non-linear.

	Accuracy Before PCA	Accuracy After PCA	Precision Before PCA	Precision After PCA	F1-Score Before PCA	F1-Score After PCA
Random Forest	0.94	0.78	0.94	0.63	0.94	0.67
SVM	0.88	0.72	0.88	0.57	0.88	0.60
Logistic Regression	0.85	0.69	0.86	0.54	0.85	0.57
KNN	0.83	0.68	0.84	0.53	0.83	0.56
Decision Tree	0.76	0.61	0.76	0.48	0.76	0.48
GaussianNB	0.55	0.40	0.62	0.30	0.54	0.27

All Results are reserved to two decimal places

Figure 17: Accuracy, Precision and F-1 Scores Comparisons Before and After PCA For Random Forest, SVM, Logistic Regression, KNN, Decision Tree, and GaussianNB

7.6 Audio Classification Task on Instrument Family (Rather than Instrument Type)

We extended our scope by incorporating instrument family classification within the TinySOL dataset. Considering not only the specific instrument but also its broader family (Brass, Keyboards, Strings, Winds), we sought to explore if the models evaluated would perform better with fewer classes but more data per class given that the family class contains all the sample audio files from the instruments from that family.

Class	Training Set	Validation Set
Brass	0.125280	0.130766
Keyboards	0.243506	0.254814
Strings	0.430699	0.428124
Winds	0.200515	0.186296

Figure 18: The ratios of each class in the training and validation sets for the Instrument Family Classification Task

For the dataset, the Brass family comprises the following instruments: bass tuba, french horn, trombone, and trumpet in C. The Keyboards family is comprised solely of the accordion instrument. The Strings family comprises Contrabass, violin, viola, and violoncello instruments. Lastly, the Winds family is comprised of the instruments bassoon, clarinet in b-flat, flute, oboe, and alto saxophone.

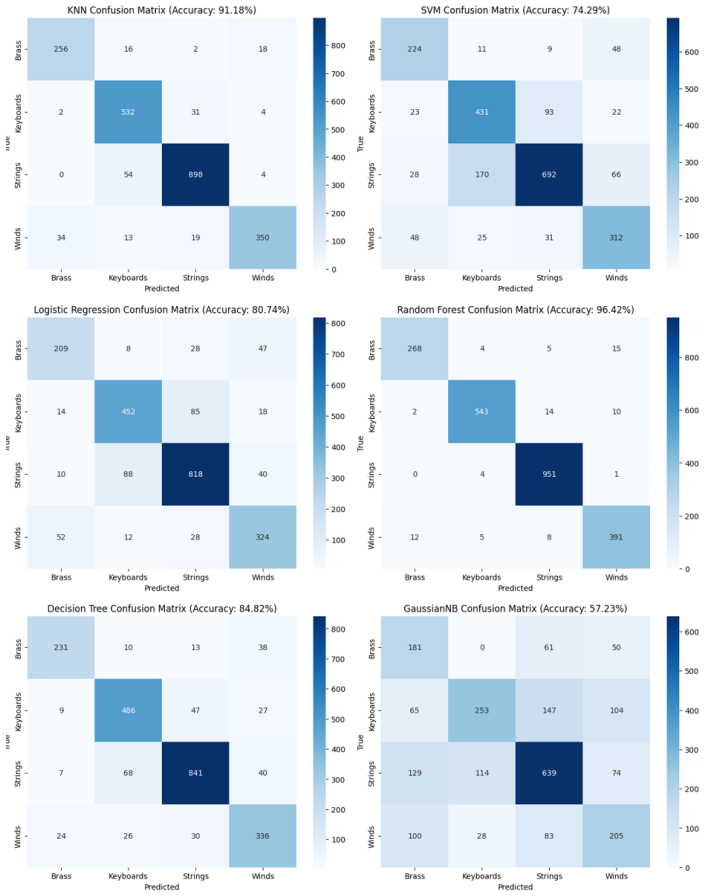

Figure 19: Confusion matrix of KNN, SVM, Logistic Regression, Random Forest, Decision Tree, and Gaussian Naive Bayes

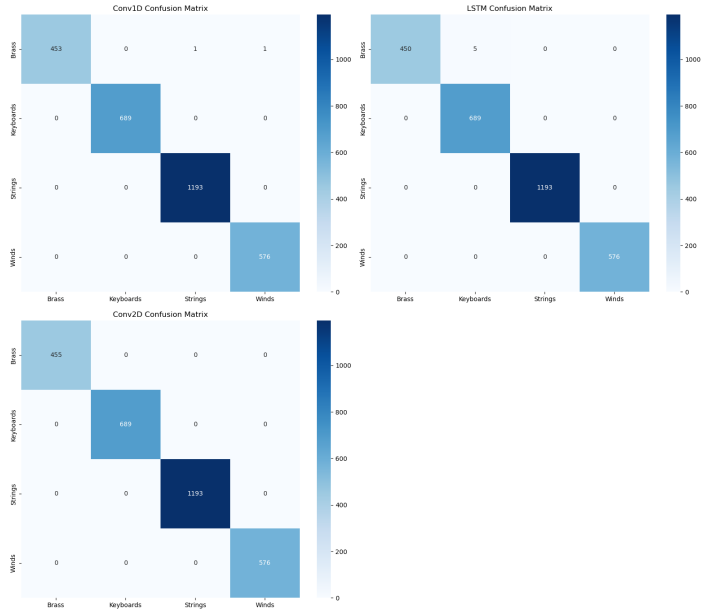

Figure 20: Confusion matrix of Conv1D, Conv2D, and LSTM

Confusion Matrix Summaries:

Model: KNN

Class	Correct	Incorrect	Samples
Brass	256	36	292
Keyboards	532	37	569
Strings	898	58	956
Winds	350	66	416

Model: Random Forest

Class	Correct	Incorrect	Samples
Brass	269	23	292
Keyboards	543	26	569
Strings	951	5	956
Winds	388	28	416

Model: SVM

Class	Correct	Incorrect	Samples
Brass	224	68	292
Keyboards	431	138	569
Strings	692	264	956
Winds	312	104	416

Model: Decision Tree

Class	Correct	Incorrect	Samples
Brass	230	62	292
Keyboards	490	79	569
Strings	847	109	956
Winds	336	80	416

Model: Logistic Regression

Class	Correct	Incorrect	Samples
Brass	209	83	292
Keyboards	452	117	569
Strings	818	138	956
Winds	324	92	416

Model: GaussianNB

Class	Correct	Incorrect	Samples
Brass	181	111	292
Keyboards	253	316	569
Strings	639	317	956
Winds	205	211	416

Figure 21: Confusion Matrix Summaries for Model Performances on Validation Set

Confusion Matrix Summaries:

Model: Conv1D

Class	Correct	Incorrect	Samples
Brass	453	2	455
Keyboards	689	0	689
Strings	1193	1	1193
Winds	576	1	576

Confusion Matrix Summaries:

Model: Conv2D

Class	Correct	Incorrect	Samples
Brass	455	0	455
Keyboards	689	0	689
Strings	1193	0	1193
Winds	576	0	576

Confusion Matrix Summaries:

Model: LSTM

Class	Correct	Incorrect	Samples
Brass	450	5	455
Keyboards	689	5	689
Strings	1193	0	1193
Winds	576	0	576

Figure 22: Confusion Matrix Summaries for Model Conv1D, Conv2D, and LSTM

	Model	Brass	Keyboards	Strings	Winds
Correct %	KNN	87.671	93.497	93.933	84.135
Error %	KNN	12.329	6.503	6.067	15.865
Correct %	SVM	76.712	75.747	72.385	75.000
Error %	SVM	23.288	24.253	27.615	25.000
Correct %	Logistic Regression	71.575	79.438	85.565	77.885
Error %	Logistic Regression	28.425	20.562	14.435	22.115
Correct %	Random Forest	91.781	95.431	99.477	93.990
Error %	Random Forest	8.219	4.569	0.523	6.010
Correct %	Decision Tree	79.110	85.413	87.971	80.769
Error %	Decision Tree	20.890	14.587	12.029	19.231
Correct %	GaussianNB	61.986	44.464	66.841	49.279
Error %	GaussianNB	38.014	55.536	33.159	50.721

Figure 23: Percentage Table of Correct and Error Rates for KNN, SVM, Logistic Regression, Random Forest, Decision Tree, and Gaussian Naive Bayes

Model Performance Metrics:

	Model	Accuracy	Precision	F1 Score
3	Random Forest	0.9632781012091357	0.9632850391074853	0.963148125542147
0	KNN	0.9117778772951187	0.9131723025444967	0.9117038563802948
4	Decision Tree	0.8522167487684729	0.8533555435210938	0.8524855320183322
2	Logistic Regression	0.8074339453649798	0.807443135554151	0.8073769098670457
1	SVM	0.7429467084639498	0.7519425108243788	0.7446265373880524
5	GaussianNB	0.5723242274966412	0.5954014719976005	0.5755419296131776

Figure 24: Model Performance Metrics for KNN, SVM, Logistic Regression, Random Forest, Decision Tree, and Gaussian Naive Bayes

CNN Model Performance Summary:

Model	Accuracy	Precision	F1 Score
Conv1D	0.9867	0.9643	0.9832
Conv2D	0.9865	0.9722	0.9839
LSTM	0.9779	0.9622	0.9783

Figure 25: Model Performance Metrics of Conv1D, Conv2D, and LSTM

Figure 26: Percentage Table of Correct and Error Rates

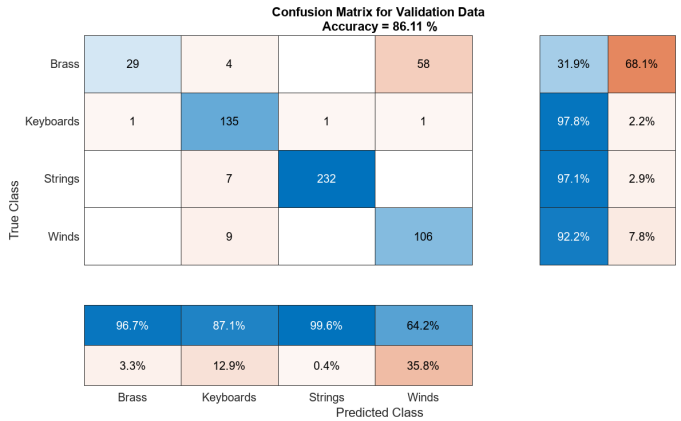

Figure 27: VGGish Confusion Matrix

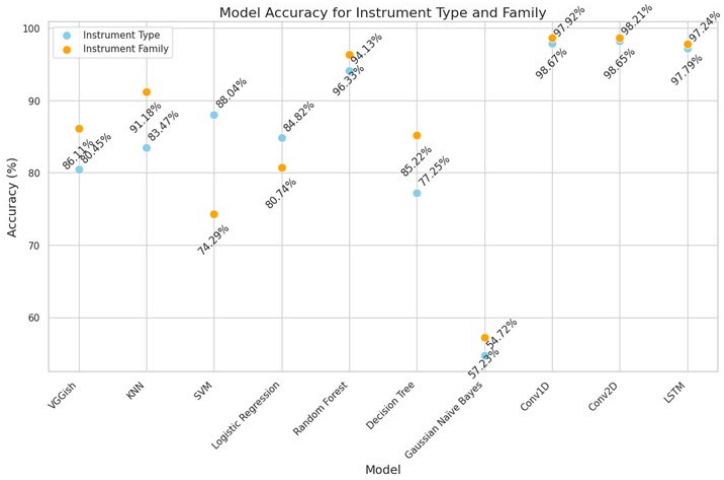

Figure 28: Model Accuracy Comparison for Instrument Type and Family

From Figures 18 - 27, we can see that model performance varies between the tasks of classifying "instrument type" and "instrument family". The comparison is summarized succinctly in Figure 27. The following observations can be made: The Convolutional Neural Network models (Conv1D, Conv2D, and LSTM) exhibit consistently high accuracy for both Instrument Type and Instrument Family classification, suggesting their robustness in capturing intricate patterns in the audio data. The Support Vector Machine (SVM) model shows a noticeable discrepancy in accuracy between the two tasks. This indicates that the SVM model might be more effective in discerning specific nuances related to Instrument Type. Gaussian Naïve Bayes, on the other hand, demonstrates a considerable decrease in accuracy for both tasks. It achieves 54.72% accuracy for Instrument Type and 57.23% for Instrument Family, implying limitations in its ability to capture complex relationships within the audio data. Random Forest is a highly effective model, achieving notably high accuracy for both tasks (94.13% for Instrument Type and 96.33% for Instrument Family). This suggests that Random Forest excels in handling the diverse and

nuanced nature of the audio dataset. The VGGish model maintains a relatively consistent performance across both tasks, showcasing its stability in capturing relevant features for both Instrument Type (80.45%) and Instrument Family (86.11%) classification.

8. DISCUSSION AND PRIOR WORK

8.1 Prior Work Revisit

The TinySOL dataset was also analyzed by Seema R. Chaudhary et al. [6], focusing on musical instrument recognition through audio features and an integrated entropy method. Their study involved two experiments. In the first experiment, the SVM model was employed to assess classification accuracy (%) by utilizing temporal and spectral features. With the integrated entropy method, they achieved an average accuracy of 60% within the same family of instruments. This was compared to the average accuracy of 50% obtained when using temporal ZCR and spectral features without integrating the entropy method. Consequently, the overall accuracy increased by 20% to 30% in familywise classification when incorporating the zero crossing rate (ZCR) and spectral features with the integrated entropy method. In the second experiment, they explored Mel-Frequency Cepstral Coefficients (MFCC) and Gammatone Frequency Cepstral Coefficients (GFCC) features in conjunction with entropy. The results showed that within the TinySOL dataset, the highest accuracy achieved was 98.52% for brass, 94.77 for string instruments, and 86.91% for wind instruments, using MFCC with the integrated entropy method. Additionally, the GFCC with integrated entropy method yielded an accuracy of 99%.

In previous work, the integration of the entropy method with features aimed to enhance classification accuracy. Our project shares a common objective with the same dataset, specifically, performing audio classification using the TinySOL dataset to achieve the highest possible accuracy. The previous work focused on comparing the accuracy of using and not using the entropy method with selected features, demonstrating how entropy integration improves classification accuracy. However, their evaluation was limited to the SVM model.In contrast, our project aimed to identify the model with the highest accuracy for audio classification. We employed ten models: VGGish, KNN, SVM, Logistic Regression, Random Forest, Decision Tree, GaussianNB, Conv1D, Conv2D, and LSTM. These models were utilized to classify audio samples from the TinySOL audio dataset into instrument types and instrument families. Comparing the accuracy achieved by the SVM model with comparable components to the one we used in our experiments, our SVM model achieved an accuracy of 0.88, surpassing the 60% accuracy achieved by Seema R. Chaudhary et al.'s SVM model. This result suggests that our decision regarding the splitting of training and validation sets, the data processing steps, and our efforts to reduce overfitting, were effective in achieving a higher level of accuracy in comparison.

8.2 Discussion

The three CNN models exhibited the highest accuracy ranging from 97% to 98%, suggesting that multi-layer architecture may be more effective for multi-class problems (rather than binary classification). Conversely, the Gaussian Naive Bayes model's performance was markedly lower, which could reflect a mismatch between the model's assumptions and the data's underlying characteristics.

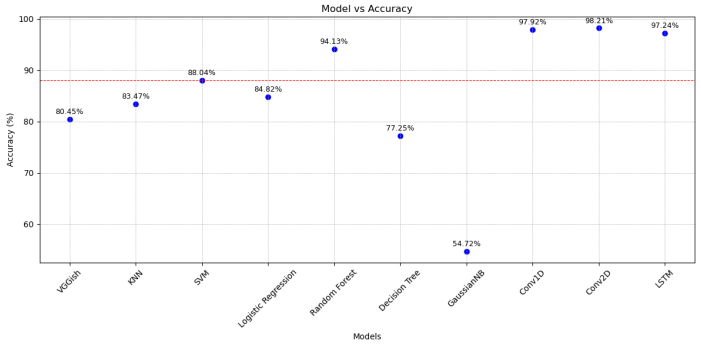

Figure 29: Model Accuracy Comparison For Instrument Types. The dotted red line denotes the SVM baseline.

In terms of baseline comparison, only the Random Forest, Conv1D, Con2D, and LSTM models surpassed the 88.04% accuracy observed on the baseline model (SVM). The accuracy of the VGG model ranged from 72% to 95% as the model was retrained on different partitions of the dataset (80% for the training set and 20% for the validation set). Part of the difference in performance can be explained by the variations in class distributions among the training and validation subsets as the data split was not explicitly balanced in the MatLab code for the VGGish model only. Additionally, some subsets might have more diverse or representative samples, leading to better generalization, while others may be more skewed or noisy, impacting model performance. Moreover, data preprocessing steps such as audio segmentation introduce undesirable perturbations to performance.

The high dimensionality of features in our dataset might also significantly impact performance. From our preliminary results, the Gaussian Naive Bayes model was the worst performer. This can be explained by the following reason: Gaussian Naive Bayes assumes that features follow a Gaussian distribution. In high-dimensional spaces, this assumption may not hold, which can affect the model's performance [5].

The outstanding performance of CNN models becomes evident when considering insights gleaned from multiple evaluation metrics, including the confusion matrix, ROC curves, F1 Scores, and Precision. The confusion matrices provide a comprehensive view, revealing the proficiency of CNN models across various classes. The preeminence of true positive counts along the diagonal underscores the models' high accuracy, indicating that, for the majority of samples, the predicted label aligns with the true label, resulting in precise classifications.

Analyzing the ROC curves further elucidates the strengths of CNN models. The high Area Under the Curve (AUC) values suggest a notable capability in capturing temporal or sequential information within the data. This aptitude is particularly advantageous in scenarios involving time series or sequence data, where CNN models excel at discerning intricate patterns, ultimately contributing to their high accuracy.

Additionally, the high F1 scores and Precision scores provide additional layers of insight into the robust performance of CNN models. These metrics affirm the models' proficiency in achieving a balance between precision and recall, crucial in scenarios where both false positives and false negatives carry significant consequences. In essence, the nuanced evaluation using multiple metrics underscores the multifaceted strengths of CNN models in handling diverse data characteristics and contributes to a comprehensive understanding of their superior performance.

Among the three CNN models, Conv2D has the highest accuracy, precision, and F1 score, indicating that it is the most accurate audio classification model and can reach an accuracy of 98.21%. While Conv1D and LSTM also performed very well, their slightly lower accuracy can be understood through their design focus. Conv1D excels in extracting features from sequence data where temporal dynamics are linear, and LSTM is particularly effective in capturing long-term dependencies in time-series data. However, if the crucial information in the audio data was more spatially than temporally distributed, this could explain why Conv2D slightly outperformed them. The nature of the TinySOL dataset might have been more aligned with the strengths of Conv2D. For instance, the dataset contains complex, multidimensional audio patterns, Conv2D's ability to analyze two-dimensional input has given it an edge in identifying subtle differences between instrument sounds more effectively than Conv1D and LSTM.

In addition to the CNN models, we attempted to apply PCA to Random Forest, SVM, Logistic Regression, KNN, Decision Tree, and GaussianNB models. The goal was to reduce the number of features in the dataset while retaining essential information, with the original purpose of increasing model accuracy. This approach was motivated by the recognition that some models are sensitive to the high dimensionality of the dataset. However, upon applying PCA to these models, we observed a decrease in the accuracy of classification rather than an improvement. Consequently, we concluded that applying PCA to our dataset to reduce dimensionality to enhance model accuracy did not yield positive results in this context. This decline in accuracy could potentially be attributed to issues such as overfitting or the loss of important information as a consequence of PCA.

9. CONCLUSION

Our study assessed ten machine learning models (VGGish, KNN, SVM, Logistic Regression, Random Forest, Decision Tree, GaussianNB, Conv1D, Conv2D, and LSTM), focusing on their efficacy in classifying instrument types and families within the TinySOL audio dataset. Our findings showed that CNN models are best suited for this particular classification task. The three CNN models evaluated (Conv1D, Conv2D, and LSTM) achieved the highest classification accuracies, ranging from 97% to 98%. Additionally, we explored the application of PCA to reduce dimensionality to improve classification accuracy. Despite not yielding the expected results, this investigation provided valuable insights. We learned that PCA may not universally enhance model performance, and its application could introduce potential issues that may hinder performance. In conclusion, our results offer significant insights into the suitability and performance of various models in audio classification tasks, highlighting the prominence of CNN for achieving high accuracy in the classification of instrument types and families within the TinySOL audio dataset.

10. REFERENCES

[1] Carmine Emanuele, Daniele Ghisi, Vincent Lostanlen, Fabien Lévy, Joshua Fineberg, & Yan Maresz. (2020). TinySOL: an audio dataset of isolated musical notes (6.0) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.3685367

[2] Ghaddar, Bissan, and Joe Naoum-Sawaya. “High Dimensional Data Classification and feature selection using support Vector Machines.” European Journal of Operational Research, vol. 265, no. 3, 2018, pp. 993–1004, https://doi.org/10.1016/j.ejor.2017.08.040

[3] Cramer, Aurora Linh, et al. “Look, listen, and learn more: Design choices for Deep Audio Embeddings.” ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, https://doi.org/10.1109/icassp.2019.8682475

[4] Hershey, Shawn, et al. “CNN architectures for large-scale audio classification.” 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, https://doi.org/10.1109/icassp.2017.7952132

[5] Djolonga, J. (2013). High-Dimensional Gaussian process bandits. https://papers.nips.cc/paper_files/paper/2013/hash/8d34201a5b85900908db6cae92723617-Abstract.html

[6] Seema R. Chaudhary, Sangeeta N. Kakarwal, R.R. Deshmukh, Musical instrument recognition using audio features with integrated entropy method, 2021. J. Integr. Sci. Technol., 2021, 9(2), 92-97. https://pubs.iscience.in/journal/index.php/jist/article/viewFile/1359/775.