Music Instrument Detection Using Gaussian Naive Bayes and Convolutional Neural Network

Theodore Matthew Patwari
Data Science
Ritsumeikan Uiversity
Kusatsu, Japan
26002104902

I. Introduction

This report will discuss the musical instrument detection, where given an audio clip, the musical instrument present in the clip will be predicted. The practical use of musical instrument detection may be automatic metadata creation. As the name suggests, metadata refers to data that describes other data. Metadata gives data some contextual information that cannot be easily extracted from that raw data itself. In the case of large amounts of data, metadata can improve searchability, which will benefit parties who are interested in using the data for various purposes. In the context of musical instrument detection, the generated metadata lets large datasets of audio clips be queryable based on its contents.

The creation of metadata through manual efforts is often laborious and expensive. Therefore, machine learning techniques, both traditional and deep learning methods, will be used to predict the musical instrument present in a given audio clip. This project includes feature engineering on audio data and classification through Gaussian Naive Bayes and Convolutional Neural Network.

II. Domain Information

Prior to performing feature engineering and machine learning, the nature of sound needs to be thoroughly understood. Sound waves contain a lot of extractable features such as pitch, amplitude, and timbre. The amplitude simply refers to how loud the sound is. Pitch refers to the frequency at which the sound waves oscillate. In music, this refers to the note that corresponds to the sound. For example, when sound oscillates at 261.63 Hz, it would be referred to as middle C (the C note at the center of a piano). Timbre, or sometimes referred to as tone color or tone quality, is another aspect of sound that gives it some distinct quality. In fact, this feature is responsible for making two instruments playing the same pitch and amplitude to still sound distinguishable.

Timbre is a factor of many things, such as the shape and material of the instrument and the technique of the player. For example, trumpet and saxophone sounds might have similar timbres since they are both made out of brass. Similarly, the guitar and harp might have similar timbres since both use nylon strings to produce music. Some instruments also possess a more diverse sound profile, due to the number of playing styles it is compatible with. To give an example, the guitar can be played with fingerstyle, chord strumming, bass plucking, or some combination of the three. On the other hand, instruments like the piano sound more consistent regardless of playing style.

In musical audio clips, different pitch, amplitudes, and timbres may be present. This is because audio clips often include sounds from varying instruments. For instance, an audio clip of a violin playing the melody of a song might include piano sounds as harmony in the background. The melody in any given audio clip is differentiable from the harmony. When a melody is isolated, it can still qualify as a song. Harmony, on the other hand, is only there to accompany the melody. In a sense, harmonies are decorative sounds. Besides the presence of other instruments, musical audio clips may also be prone to environmental noise, especially if they are not recorded in a sound-proof studio. For example, an audio clip of someone playing music in public may include talking or footstep sounds.

III. Data Acquisition and Analysis

Before delving into the project itself, the data collection process will be briefly discussed. The dataset used to perform the classification task was scraped from YouTube. YouTube was a suitable source of data due to the abundance of music cover videos that can be searched easily. Initially, seven musical instruments were selected as the classes. These instruments are piano, saxophone, trumpet, violin, flute, guitar, and harp. Then, a script was created to download YouTube videos that were returned given the following query format: "[musical instrument name] covers". In total, about 100 videos for each musical instrument were downloaded by the scraping script. Afterwards, the downloaded videos were broken into smaller chunks of five second audio clips. From here, 800 audio clips were retrieved for each musical instrument. These 800 clips did not utilize all 100 videos. This was a deliberate decision after taking limited storage and computation power into consideration (since the process of feature engineering the audio clips require significant computation power). All audio clips are formatted as .mp3 files.

Some analysis shows that the data was met with some limitations. Since the data are extracted from music covers on YouTube, the data contains some noise. Instrument covers on YouTube may contain other instrument sounds in the

background as well. For example, a saxophonist may play the melody of the song while piano and drum accompaniments are audible in the background. Besides that, some YouTube videos include moments where the instrument is not playing. The start of the musical cover videos may be completely quiet or solely feature a different musical instruments. To give a specific example, one saxophone cover in the dataset featured a lengthy piano intro. To get around this, the first minute of a YouTube video is skipped over before cutting the video into clips.

IV. Feature Engineering

A. Loading the Data

Feature engineering heavily utilizes the Librosa library for Python, which allows complicated operations to be done easily. The downloaded audio clips were first loaded in and converted into floating point time series. A sampling rate was defined to determine how many samples are taken from the audio file every second. Here, Librosa's default value of 22050 was used, meaning that the total number of samples taken and the output length of each audio clip is 110250 (22050x5).

B. Background Noise Filtering

The background noise mentioned in the previous section was removed through filtering. What filtering essentially does is that it recognizes certain frequencies that contribute to background noise and suppresses them. Specifically, frequencies that belong to the melodic component of the music are isolated. The way melody, or foreground, is distinguished from background audio is through the sparsity of frequencies. It can be assumed that the frequency distribution of melodic components are more sporadic, while that of harmonic components tend to be more consistent over time.

Filtering utilizes a function called Fourier Transform. This function deconstructs sound into the frequencies that make it up and shows the magnitude that individual frequencies have in creating the sound. Here, Short-time Fourier Transform (STFT), was used. STFT is a variant of Fourier Transform that utilizes an overlapping sliding window and performs the transformation on small chunks of the data stream.

The magnitude information after Fourier Transformation on the audio clip was used to create a filter that can isolate foreground musical instrument sound through nearest-neighbor aggregation. Nearest-neighbor aggregation was done by chunking the magnitude data into frames. From here, each frame is assigned a set of other frames that are proximate to it by calculating cosine similarity. Then, for every frame, the median of similar frames are taken and appended to a new series of frames. The median operation was used in order to suppress any sporadicity in the values. The end result of these operations is a new series of frames of equal length to the magnitude frames.

Next, the minimum frame between the nearest-neighbor aggregated frames and the original magnitude frames are taken to create a filter. The filter gets normalized by dividing it over the original magnitude frames. As a result, the final filter frames consist of values between zero and one.

This filter can be used as a mask such that values close to one correspond to background noise. A softmasking operation was done to keep frames from the original magnitude data that correspond to a low value in the mask, resulting in a series of magnitude frames from the foreground sound. The resulting foreground frames are then passed to an Inverse Short-time Fourier Transform (ISFT) function, reconstructing it as human-audible sound.

C. Mel Spectrogram Generation

The next feature engineering task is to generate mel spectrogram images from the filtered audio. Spectrograms refer to a visualization of frequencies and amplitude over time. In this project, the x and y axis of the spectrogram refer to time and frequency respectively. The color of each pixel in the spectrogram image refers to the amplitude of a particular frequency at one point in time.

The mel spectrogram is a type of spectrogram that uses the mel scale. In the Hertz scale (which is typically used to represent frequency), equal frequency intervals might not be perceived as equidistant to human listeners. This is because human hearing is less sensitive in higher frequency ranges. The mel scale fixes this issue by using a logarithmic transformation to frequency values.

Mel spectrogram generation uses mel-banks, which are used to bin Fourier Transform magnitudes by applying triangular filters. Triangular filters are used to calculate the weighted average of the values in the span of the triangle's base. The number of mel-banks used correspond to the number of triangles used and is directly proportional to the amount of details of the generated mel spectrogram. In this project, 128 mel-banks are used on the filtered audio. Furthermore, 8000 Hz was selected as the maximum frequency, thus ignoring higher frequencies when generating mel-banks.

After the mel spectrograms were made for every filtered audio clip, the generated image was converted into grayscale. This was done to reduce the image to a single channel (as opposed to three channels when using RGB images), effectively making the image smaller and computationally cheaper to work with.

D. Mel Spectrogram Image Processing

Prior to feeding the spectrogram image into machine learning models, some preprocessing steps were done. Firstly, the spectrogram gets cropped. This was done because with the way the spectrogram images were saved, a white border appears around the image. These white borders were cropped off in order to make sure that the spectrogram that enters the model only contains useful data for learning. It was also done to minimize the size of the input as much as possible with the goal of minimizing the computational requirements for training. Next, the image was reshaped in order to meet the shape requirements of the libraries used for building the deep learning model. Specifically, the model requires the first

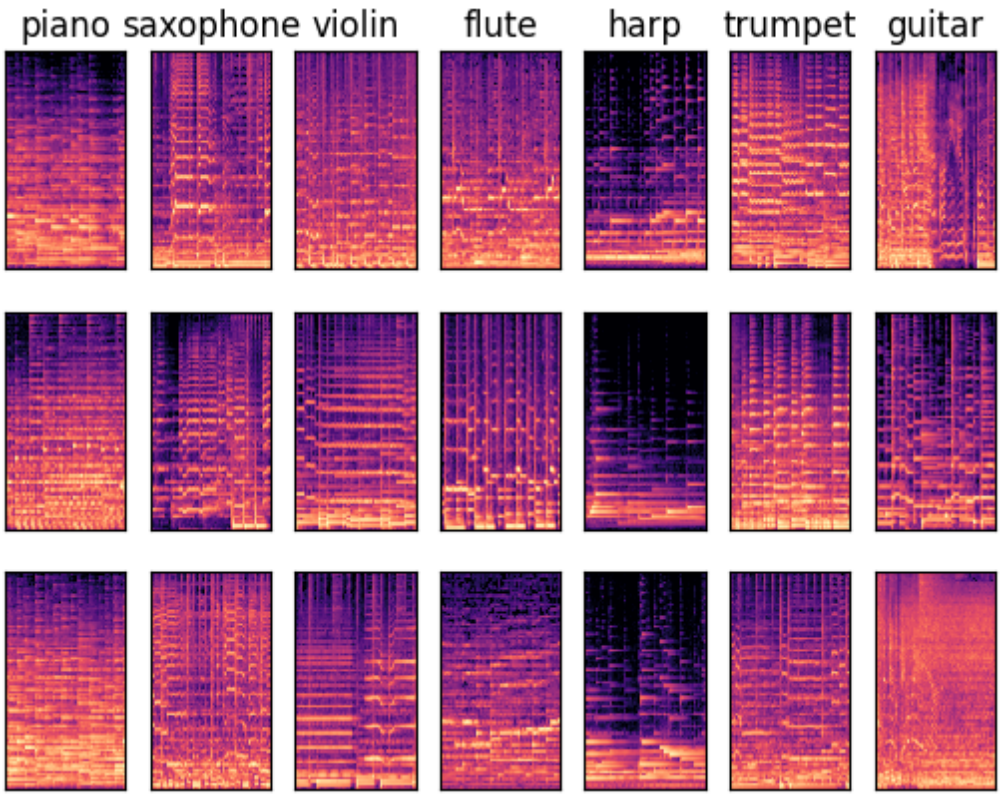

Fig. 1. Unfiltered mel spectrogram of all instruments

dimension to represent the number of channels, while the succeeding dimensions represent the width and height of the image. Then, the image gets normalized, such that all the pixel values are between zero and one. The image was also flattened when being used for training the Gaussian Naive Bayes model (but not for deep learning), since the model only accepts one dimensional data.

Examples of mel spectrograms of the raw (unfiltered) audio clips from the dataset can be seen in Figure 1. Visually, it shows how the spectrograms of different instruments possess some defining characteristics. For example, the spectrogram of violins tends to have long stretches of yellow lines, while flute spectrograms have sparse appearances of yellow points.

Figure 2 shows an example of the mel spectrogram of filtered audio after conversion to grayscale and normalization. Here, the spectrogram contains more black pixels since filtering loses most of the contents of the audio clips. The colors of the pixels are also muted due to the effect of normalization.

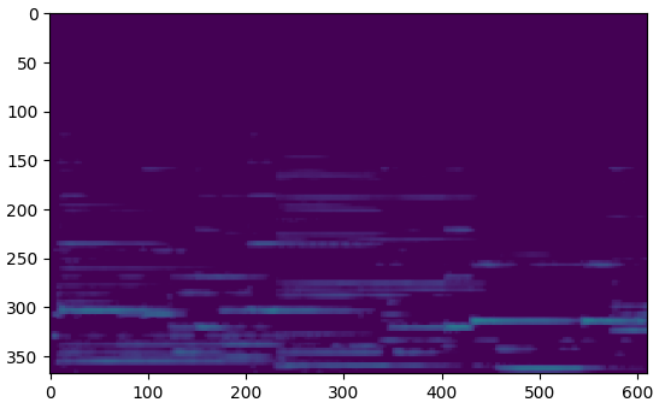

Fig. 2. Mel spectrogram of audio from the training set after filtering, normalization, cropping, and grayscaling

V. Machine Learning Experiments

A. Project Setup

This musical instrument classification task was done using Python, utilizing libraries like PyTorch (for building and training deep learning models), Librosa (for feature engineering audio data), and Scikit-Learn (for building and training traditional machine learning models). In terms of hardware, the A100 Cloud GPU provided by Google through Google Colab was used.

B. Gaussian Naive Bayes

A Gaussian Naive Bayes model was used to perform classification on the mel spectrogram images. A Naive Bayes model predicts the probability of a sample belonging to a class given a set of features. Given each feature, the probability of

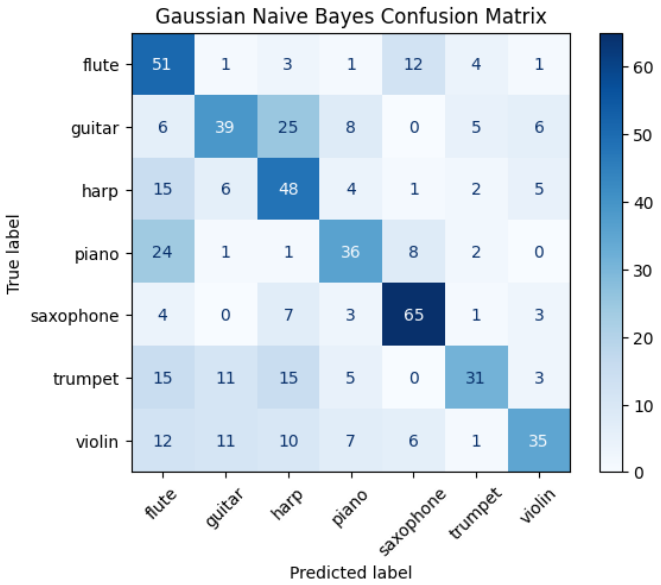

Fig. 3. Confusion matrix from the Gaussian Naive Bayes model

TABLE I
PRECISION, RECALL, F1, AND SUPPORT FROM GAUSSIAN NAIVE BAYES PREDICTIONS

	precision	recall	f1-score	support
flute	0.64	0.56	0.60	82
guitar	0.53	0.38	0.44	66
harp	0.73	0.80	0.76	91
piano	0.52	0.71	0.60	84
saxophone	0.45	0.40	0.43	72
trumpet	0.67	0.34	0.45	88
violin	0.44	0.66	0.53	77
accuracy			0.56	560
macro avg	0.57	0.55	0.54	560
weighted avg	0.58	0.56	0.55	560

a class is calculated given that feature. This value is known as the likelihood probability. In the case of image classification, each pixel is considered a feature. There are many variations of Naive Bayes, each assuming a different distribution of features. Gaussian Naive Bayes, as the name suggests, assumes that the pixels are normally distributed.

The Naive Bayes model was trained through partial fitting on smaller batches instead of on the entire dataset at once. This was done in order to make the training process computationally efficient. A batch size of 32 was arbitrarily selected, meaning that the model will train on 32 data samples per iteration. The trained model was evaluated by introducing unseen test data, sampled with a batch size of 16. The result shows 56% accuracy, with a baseline accuracy of 14% (when the model does random guessing, the accuracy should be at about 14%). Figure 3 shows the confusion matrix produced by the Gaussian Naive Bayes model. Table 1 shows the precision, recall, and support values from the Naive Bayes model.

C. Convolutional Neural Network

A deep learning approach using Convolutional Neural Network (CNN) to this music instrument classification task was also done. CNNs were selected due to its ability to take spatial relationships into account. Hence, CNNs are useful for image related tasks, where the semantic meaning of images depends on the specific arrangement of pixels in two dimensional space. The CNN architecture in this project includes two convolution layers and two dense layers, as shown in Figure 4.

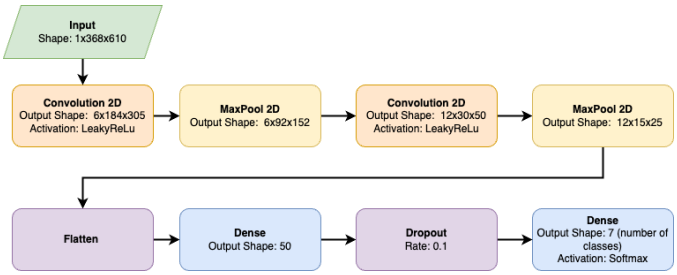

Fig. 4. CNN architecture

Since the initial image is in grayscale, it only has one channel. Increasing the number of channels helps the model elicit useful features. Hence, the two convolution layers turn their inputs to 6 and 12 channels respectively. They also decrease the input dimensions. The two dense layers simply decrease the length of the input until it matches the number of classes. Finally, a softmax activation was used to transform the output into the probabilities of the input belonging to each class.

Some experiments were done with the model architecture. When a third convolution layer was added, the validation accuracy of the model seemed to decrease by about 5%. Adding a third linear layer also yielded similar results. However, when a third convolution layer and a third linear layer are used simultaneously, the validation accuracy drops to about 14%. Here, the validation accuracy matches the baseline accuracy, meaning that the model's predictive ability matches that of random guessing.

At each training iteration, the loss and accuracy on training and validation data were computed. The validation data do not include any training data, and validation losses are not used for backpropagation. The training process was seeded for reproducibility.

Since this is a multiclass classification task, the cross entropy loss was utilized. The model uses the Adam optimizer with a learning rate of 1e-3. The optimizer and learning rate value was selected after iterative experimentation with alternatives. AdamW, RMSprop, and SGD were at one point used for training. Altering the optimizers leads the model to have similar accuracies, however, the Adam optimizer was able to lead the model to converge faster. Similar experiments were done with the learning rate. After trying higher learning rates, 1e-2 and 5e-2, as well as lower rates, 1e-4 and 5e-4, it was found that 1e-3 was the most suitable value. This is because higher learning rate caused the accuracy of the model to reach

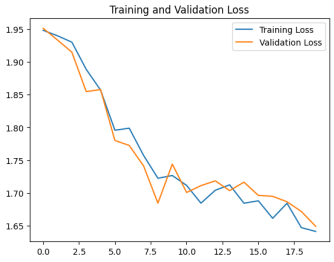

Fig. 5. Training and validation loss

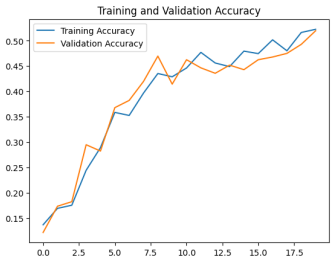

Fig. 6. Training and validation accuracies

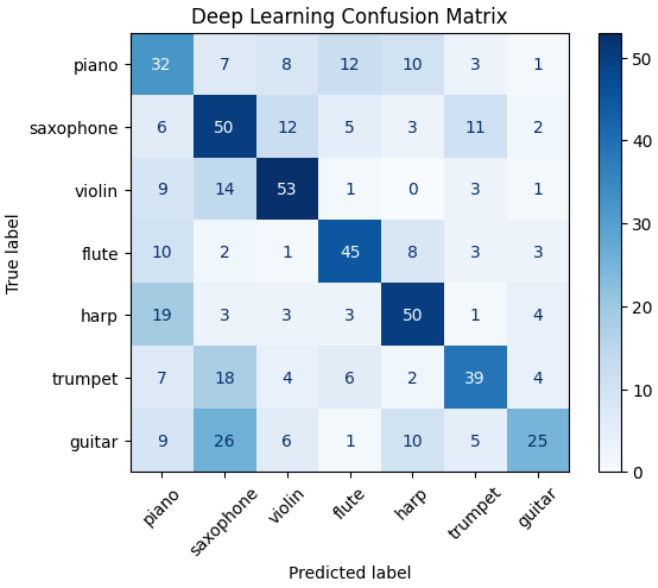

Fig. 7. Confusion matrix from the CNN model

a plateau on some local minima (when validation accuracy was around 27%). On the other hand, lower learning rates caused the model to take smaller loss improvements, eliciting long training times.

Besides that, experiments with learning rate schedulers were also done. Schedulers were used to decrease the learning rate during training time. Firstly, the multistep scheduler was implemented, where a specific epoch was defined during which the learning rate will be decreased. Second, a reduce on plateau scheduler was implemented, where the learning rate gets decreased when a defined metric stays stagnant for some period of time. Reduce on plateau scheduler was experimented on with two metrics: validation accuracy and validation loss. When the validation loss stops increasing after five epochs, the learning rate gets decreased. Such is the case for when the validation loss stops decreasing. However both learning rates have made the model converge slower. Empirical experimentation with the learning rate suggests that a static learning rate of 1e-3 is best for the developed model.

The training data was sampled with a batch size of 32 while the validation data was sampled with a batch size of 16. Training was done with 35 epochs. The trained model shows a validation accuracy of 52% with a baseline accuracy of 14%. Figure 5 and 6 respectively show the progression of loss and accuracy of the model over 20 epochs (on training and validation data). Figure 7 shows the confusion matrix produced by the CNN model.

VI. Discussion and Conclusion

Both the Gaussian Naive Bayes and Convolutional Neural Network models yield similar prediction accuracies on unseen data. These accuracy values are well above the baseline accuracy of 14%, indicating that the model was in fact able to learn some distinguishing qualities between different instrument sounds. However, an accuracy of around 50% would still need to face improvements before being practically useful.

TABLE II
Precision, recall, f1, and support from CNN predictions

	precision	recall	f1-score	support
flute	0.69	0.67	0.68	82
guitar	0.39	0.23	0.29	66
harp	0.56	0.53	0.55	91
piano	0.50	0.61	0.55	84
saxophone	0.35	0.50	0.41	72
trumpet	0.56	0.44	0.49	88
violin	0.56	0.60	0.58	77
accuracy			0.52	560
macro avg	0.52	0.51	0.51	560
weighted avg	0.52	0.52	0.51	560

The confusion matrix between the two models shows similar results. Both models were able to classify piano sounds without much misclassification. On the other hand, the saxophone and violin sounds are often misclassified. The Naive Bayes model misclassified saxophone sound as violin, and the CNN misclassified violin sound as saxophone, despite how the two instruments belong to different instrument families. However, looking at the mel spectrograms for the two instruments, some similarities arise. At hindsight, such results might seem perplexing, however upon observing the mel spectrograms from Figure 1 of the two instruments, some similarities can be observed. It can be seen that the two mel spectrograms consist of wavy activations of yellow areas. This shows rapid oscillations of pitch, known as *vibrato*. The technique is often employed in many string and wind instruments like the violin and saxophone, which may cause the models to mix up the two. This may also be the reason why the Naive Bayes misclassified some trumpet samples as violins and the CNN

misclassified trumpet samples as saxophones, as the trumpet also uses *vibrato* a lot.

The Naive Bayes model performed well on harp samples. The CNN displayed good performance as well, although there were some misclassifications into the piano and guitar classes. This misclassification may have happened since the harp, piano, and guitar all belong to the string family. Furthermore, both the harp and guitar utilize nylon as some of their strings, resulting in identical timbral qualities.

Perhaps, the most notable result is how both models displayed poor capabilities in classifying guitar sounds. This is reflected in two two confusion matrices. Guitar predictions received a low precision value in the CNN model and a low recall value the Naive Bayes model, as seen in Tables 1 and 2. This may be attributed to the diversity of guitar sounds. As described in Section 2, the guitar is a versatile instrument with many ways to play. Besides that, both acoustic and electric guitars produce different sounds, and are both labeled under the same "guitar" label in the dataset. Due to this, the trained models might have difficulties in generalizing the spectrogram qualities of guitars.