# CLASSIFYING MUSICAL INSTRUMENTS THROUGH NEURAL APPROACHES: AN EMPIRICAL STUDY

**JUHYOUNG PARK**
**Bachelor of Science, University of Lethbridge, 2024**
**Bachelor of Science in Engineering, Kookmin University, 2009**

A thesis submitted
in partial fulfilment of the requirements for the degree of

**MASTER OF SCIENCE**

in

**COMPUTER SCIENCE**

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

CLASSIFYING MUSICAL INSTRUMENTS THROUGH NEURAL APPROACHES: AN EMPIRICAL STUDY

JUHYOUNG PARK

Date of Defence: April 22, 2026

| | | |
|---|---|---|
| Dr. John Zhang<br>Thesis Supervisor | Associate Professor | Ph.D. |
| Dr. Wendy Osborn<br>Committee Member | Associate Professor | Ph.D. |
| Dr. Yllias Chali<br>Committee Member | Professor | Ph.D. |
| Dr. Andrew Fiori<br>Chair, Thesis Examination Committee | Associate Professor | Ph.D. |

# Dedication

To my Lord, for His guidance, grace, and unfailing faithfulness, and to my family, whose love and prayers have sustained me on this journey.

# Abstract

Musical instrument classification is one of the important tasks in Music Information Retrieval (MIR), yet achieving robust performance in real-world music is still challenging. In this thesis, we investigate the problem of multi-class, multi-label musical instrument classification through neural approaches.

Our study utilizes multi-genre instrument mixtures derived from the MUSDB18 and the MedleyDB, two popular datasets in MIR, and uses Mel-spectrogram, Mel-frequency cepstral coefficients (MFCCs), Constant-Q transform (CQT), Chroma Energy Normalized Statistics (CENS), and zero-crossing rate as audio features. Principal Component Analysis (PCA), Incremental PCA, and upsampling techniques are also employed to facilitate our experiments.

In our investigation, we have found that the simple models using Artificial Neural Networks (ANNs) show lower performance in classifying mixed-instrument classes, and the hierarchical models using multiple simple ANN-based models show slightly improved performance. The models using Convolutional Neural Networks (CNNs) outperformed the models using ANNs, and employing combined audio feature images as input to the CNN-based models improves the performance on the mixed-instruments classes. We have designed and conducted a series of empirical experiments using our proposed neural architectures on the two datasets. The results are evaluated and discussed. We expect that our approach would achieve better performance in the real-world situation.

# Acknowledgments

I am profoundly and sincerely grateful to Dr. John Zhang for his unwavering support, insightful guidance, and constant encouragement, which extended beyond this thesis to invaluable lessons in life and learning. His clear and thoughtful advice consistently illuminated the path forward, allowing this research to progress steadily through each stage.

I sincerely thank the members of my thesis committee, Dr. Wendy Osborn and Dr. Yllias Chali, for their encouragement and invaluable feedback. Their courses and academic insights have been a continual source of inspiration throughout my studies. I would also like to thank Dr. Andrew Fiori for serving as the chair of the examination committee.

I gratefully acknowledge the financial support provided by the University of Lethbridge School of Graduate Studies and Alberta Innovates, whose generous grants have contributed greatly to my research and the acquisition of essential equipment.

I deeply appreciate my colleagues for the many stimulating discussions, the exchange of ideas, and the mutual encouragement we shared, as well as for the joyful moments beyond academics.

Lastly, I would like to express my heartfelt gratitude to my family for their prayers, unwavering love, and endless support. Although we have been far apart, their faith in me has given me the strength to reach this point. Above all, I thank God for making all of this possible and for guiding me throughout this journey — in the past, present, and the days to come.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Artificial Intelligence (AI) is changing the world. People use AI for more than just curiosity. It is already changing many of our activities, including repetitive processing, human resource management, and customer service, to improve workflow efficiency [59]. The integration of AI within the field of Music Information Retrieval (MIR) represents a significant development of such a kind.

## 1.1 Music Information Retrieval

Music information retrieval (MIR) is a popular research area for academic and industrial researchers. It provides the music information needed for a variety of purposes, such as classifications of music genre or musical instrument, and music recommendation systems [34]. MIR primarily uses content-based methods, which means it extracts information from music using computational processes, including data mining and machine learning techniques. It has also evolved as technology has advanced [9, 34].

MIR uses many features extracted from music [9]. Spectral features are the most popular audio features. They represent audio signals in the time–frequency domain, which facilitates the characterization of timbre. Its popularity is driven by speech recognition research and applications. Pitch features map frequency values (Hz) to discrete pitch classes, representing sounds as musical pitches rather than raw frequencies. A scale consisting of twelve distinct pitch classes is most common in contemporary music. The interval between two pitches is a key aspect of harmony features. Temporal features represent characteristics

of audio that change over time, such as a set of single values obtained by converting short samples or frames. Rhythm features relate to the tempo, beats, and rhythmic patterns of music. Other features, such as lyrics, scores, and timesheets, provide information that is not directly extracted from audio data [14, 34]. The theoretical details concerning the audio features utilized in this thesis are detailed in Section 2.2.

MIR has a variety of applications, including generative AI for music and speech, music genre classification, musical instrument classification, music recommendation systems, mood and emotion detection, and music source separation [47]. Their preprocessing and feature extraction steps are similar. However, they address different problems, and the ways in which they utilize audio features to solve them also differ [9, 34].

## 1.2 Musical Instrument Classification

Musical instrument classification is a significant area of research in MIR. Yet, achieving high performance remains very challenging [34]. Extracting specific target information from it is challenging because music consists of multiple attributes that affect each other, such as pitch, harmonic, temporal, and timbral features that collectively express complex cultural meanings [14].

Earlier studies mainly addressed monophonic instrument classification by employing audio features such as timbre, spectrogram, tempo, and pitch, together with statistical and traditional machine learning methods including K-Nearest Neighbours (K-NNs), Naive Bayes classifiers, Decision Trees, and Support Vector Machines (SVMs) [25]. However, these methods require significant efforts to analyze audio signals and also demand supervision from audio experts, because they can easily miss important musical information and do not work well when the dataset is huge and diverse [13].

As a recent development, the application of deep neural networks to musical instrument classification overcomes these limitations by learning complex patterns directly from audio signals and reducing the time for feature extraction with costly computations. It is also

more flexible and scalable for handling larger datasets and new genres of music [13, 25].

Some widely used methods include Convolutional Neural Networks (CNNs), which primarily utilize spectrograms; Recurrent Neural Networks (RNNs), which are effective for modeling rhythm and note sequences; hybrid architectures such as Convolutional Recurrent Neural Networks (CRNNs), which capture both spatial and temporal patterns; and attention-based models, which focus on the most informative parts of the music rather than treating the entire audio signal equaly [13].

## 1.3 Neural Networks

The theory of AI has been around for a very long time. The idea of intelligent machines emerged in the $19^{th}$ century. As time passed, the perceptron model, a fundamental component of artificial neural networks, was implemented in the 1950s [49]. However, it turned out that a single perceptron was unable to solve complicated problems, and people did not know how to train models with multiple layers of perceptrons [1, 19]. People had also attempted to use specific rules to implement expert systems with AI; however, this approach was limited because it could not operate outside of these rules [12].

Substantial progress in training multi-layer neural networks was achieved when the backpropagation algorithm was introduced in 1986 [50]. Backpropagation is used to calculate gradients, which are used to update the weights in each layer. For instance, LeNet-5 was developed to classify handwritten digits using backpropagation and CNNs in 1998 [33]. However, due to issues such as vanishing gradients, local minima, overfitting, high cost of computing, and the rise of Support Vector Machines (SVMs) and Decision Trees, neural networks research became unpopular [12, 19].

In the mid-2000s, it became popular again. Remarkable research, such as Geoffrey Hinton's research about deep networks [26] drove its development, and deep neural networks became so popular because of AlexNet [31], the winner of the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition. The application of deep neural net-

works has been successful in a variety of fields. This innovation has been driven by big data, advances in computing power, including GPUs, and improvements in algorithms [1, 19].

## 1.4 Problem Statement

In our research, we plan to take the following challenges in musical instrument classification:

- **Multi-class, Multi-label Classification**: Many related studies have focused on the classification of single instruments or isolated instrument tracks [4, 51], which is hard to apply in real-world situations, where multiple instruments often overlap, and additional unwanted signals, such as background noise, are present. Multi-class classification refers to predicting a single category from multiple possible classes, and multi-label classification allows a prediction to belong to more than one category [12].

- **Classification in Diverse Musical Genres**: Many existing methods are biased toward the classification of classical instruments or instruments from specific geographic regions [7, 36, 51], which limits their ability for musical instrument classification in contemporary music. This limitation is further exacerbated by the lack of datasets that adequately cover a wide range of musical genres.

This thesis addresses these challenges by investigating multi-class, multi-label musical instrument classification in practical mixed audio data.

## 1.5 Our Contributions

In this thesis, our main contributions are as follows:

- We construct synthetic multi-genre instrument mixtures from existing individual tracks in datasets, including both classical and popular music, to support multi-class, multi-label musical instrument classification in realistic mixed audio conditions.

- We investigate multi-class, multi-label instrument classification models based on Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs), whose architectures range from simple feedforward networks to hierarchical structures, using diverse combinations of audio features.

- We provide a comprehensive experimental evaluation and analysis that reveals the effectiveness and limitations of the proposed approaches.

## 1.6 Thesis Outline

The remaining parts of this thesis are organized into five chapters. Chapter 2 provides background on neural networks and describes the five key audio features used in this thesis. It also reviews related work, ranging from statistical and traditional machine learning methods to deep learning approaches.

Chapter 3 introduces the MUSDB18 and MedleyDB datasets used throughout this thesis. It then describes the preprocessing stages, including audio preprocessing, feature extraction, and audio feature dimensionality reduction and upsampling.

Chapter 4 presents experiments on the datasets using artificial neural networks. It details the evaluation metrics and dataset-specific processing, followed by experimental results for a simple neural network and hierarchical models on each dataset from Chapter 3.

Chapter 5 presents experimental evaluations using convolutional neural networks. It describes the model architecture, the conversion of audio features into images, and the proposed dB-MFCCs. It then reports the experimental results, including the performance of different combinations of audio feature images.

Finally, Chapter 6 summarizes the contributions of our work and provides future work.

# Chapter 2

# Background and Related Works

This chapter will provide the theoretical background for neural networks and audio features for musical instrument classification, and introduce key concepts relevant to the subsequent chapters. Among popular neural network architectures, we will focus on Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs). We also provide evaluation metrics used for neural networks, such as Accuracy, Precision, Recall, and F1-score.

Audio features are essential for representing audio signals as inputs to neural networks in Music Information Retrieval (MIR) research, including musical instrument classification. We will introduce audio features that we employed, which have shown good performance in many studies, and then present the Principal Component Analysis (PCA), used to reduce the dimensionality of the data.

Finally, the chapter will conclude with a review of the literature on musical instrument classification, in which a variety of approaches are discussed.

## 2.1 Neural Networks

Neural networks are widely used in MIR research. In particular, for musical instrument classification, ANNs have been used with other machine learning methods, such as Support Vector Machine (SVM) [56], and CNNs have been applied with some adaptations [4, 63].

### 2.1.1 Artifical Neural Networks

Artificial Neural Networks (ANNs) are simplified computational network models inspired by biological neural networks. Neuroscience research has provided helpful resources

for studying neural networks, but their application to improve ANNs is still limited because we do not yet fully understand how neurons interact with each other [19].

A typical multi-layer neural network architecture is shown in Figure 2.1. ANNs use weights to represent the connections between nodes, just as synapses represent the connections between biological neurons. These weights are used to control the scaling of the input values at each layer of the network, and ANNs learn patterns from pairs of input and output data by updating the weights [1].

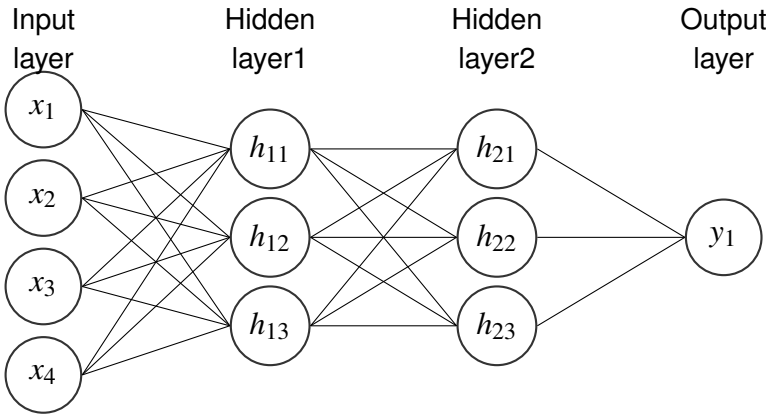



Figure 2.1: A typical multi-layer neural network architecture.

### 2.1.2 Forward Propagation

Forward propagation is performed from the input to the output layer. Let $\overline{X}_i$ be a column vector of the input layer, $\overline{X}_i = [x_1 \dots x_d]$, where $d$ is a number of nodes, and $\overline{W}_i$ be a column vector of edges of weight, $\overline{W}_i = [w_1 \dots w_d]$. The node output $\hat{y}$ is calculated by the following equation [1]:

$$\hat{y} = \Phi(\overline{W} \cdot \overline{X} + b) = \Phi\left(\sum_{i=1}^{d} w_i x_i + b\right), \tag{2.1}$$

where $\Phi$ represents the activation function, and $b$ denotes a bias. This equation is applied to every node in the neural network.

### Activation Functions

Activation functions play a crucial role in enhancing the learning ability of multi-layer neural networks [1]. Nonlinear activation functions, in particular, provide additional modeling capabilities beyond those of linear models. If all layers employ a linear activation function, the output of the network is equivalent to the output of a single-layer neural network. For example, if $f(x)$ and $g(x)$ are linear functions, their composite function $f(g(x))$ is also linear [1, 19]. The most popular activation functions are listed below:

$$\Phi(x) = \text{sign}(x) \qquad \text{(sign function)}, \tag{2.2}$$

$$\Phi(x) = \frac{1}{1+e^{-x}} \qquad \text{(sigmoid function)}, \tag{2.3}$$

$$\Phi(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad \text{(tanh function), and} \tag{2.4}$$

$$\Phi(x) = \max\{x, 0\} \qquad \text{(ReLU function)}. \tag{2.5}$$

Choosing an appropriate activation for the nodes in the output layer is important. The sigmoid function is typically used when the output value lies between 0 and 1, whereas for multi-class classification, the softmax function is used, as shown below:

$$\Phi(\overline{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{k} e^{v_j}}, \quad \forall i \in \{1, \ldots, k\} \qquad \text{(softmax function)}, \tag{2.6}$$

where $k$ is the number of output nodes [1].

### Loss Functions

Loss functions calculate the difference between the true value $y$ and the prediction $\hat{y}$ and are used to train neural networks to minimize the difference [1]. They are expressed as follows:

$$Minimize_{\overline{W}} L = L(y, \hat{y}). \tag{2.7}$$

Loss functions are chosen according to the particular problem one deals with. Mean

Squared Error (MSE) is commonly used for regression problems. Binary cross-entropy is used with the sigmoid activation function. Categorical cross-entropy is used with the softmax activation function [1, 22].

**Hamming Loss**

Hamming loss is useful when the label is represented as a binary vector. It first computes the Hamming distance between the true and predicted vectors by comparing the value at each position, and then averages the distance over all samples [22, 62].

Given an $i$-th training example $x_i$ and its true label vector $y_i$, the Hamming loss is defined as:

$$HammingLoss = \frac{1}{c}\sum_{j=1}^{c}\mathbf{1}\left[t\left(f_j(x_i)\right) \neq y_{ij}\right], \tag{2.8}$$

where $j$ indexes the label dimension, $c$ is the number of labels, $f_j(x_i)$ denotes the $j$-th component of the prediction vector produced by the model for $x_i$, $t(\cdot)$ is a thresholding function that maps scores to binary labels, and $1[\cdot]$ is the indicator function that returns 1 if the condition inside the brackets is true and 0 otherwise [62].

### 2.1.3 Gradient Descent

Neural networks learn by using backpropagation to compute gradients from the output layer back to the input layer and by using an optimization algorithm such as gradient descent to update the weights [50]. Gradient descent is expressed as follows:

$$\overline{W}_{new} \Leftarrow \overline{W}_{old} - \alpha\frac{\partial L}{\partial \overline{W}}, \tag{2.9}$$

where the gradient $\Delta L$ is expressed as $\frac{\partial L}{\partial \overline{W}}$, and $\alpha$ represents the learning rate [1]. The gradient is computed using the chain rule for the weights in multiple layers [1, 50].

The learning rate must be chosen carefully. If it is too small, training progresses very slowly, and the model may become trapped in a local minimum. If it is too large, the model may miss the optimal minimum, overshoot the target, or even cause the loss to diverge

[1, 22].

To achieve successful learning, Stochastic Gradient Descent (SGD) or mini-batch SGD is typically employed. To adjust the learning rate during training, optimizers such as AdaGrad, RMSProp, or AdaDelta can be used, while Adam (Adaptive Moment Estimation) [30] is currently one of the most widely used optimizers [1].

### 2.1.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) are neural networks that are particularly well-suited for data with a grid-like structure [19]. They are most commonly used with 2D image data, but can also be applied to time-based data, such as sequences of sensor measurements. Figure 2.2 illustrates a typical convolutional neural network architecture. CNNs are named after the convolutional operation on which they are based. In a typical CNN, the convolutional operations are followed by a ReLU activation and a pooling operation. After that, the output is passed to a fully connected neural network [1, 19].

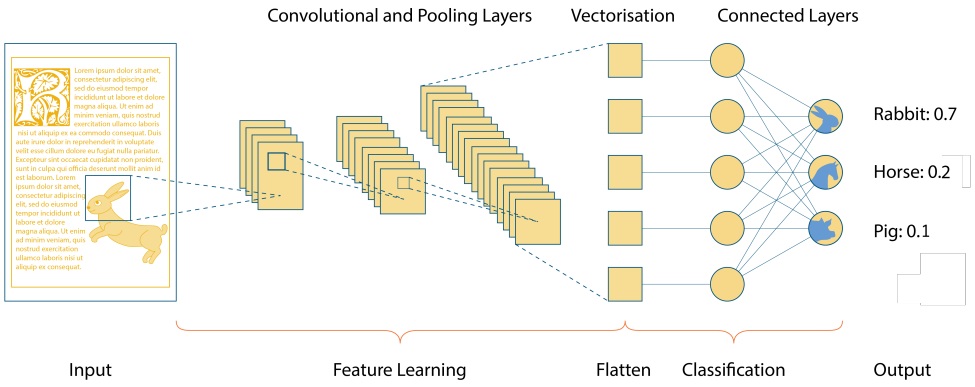



Figure 2.2: A diagram of a convolutional neural network (CNN) architecture, by Irisbox, via Wikimedia Commons, licensed under CC BY 4.0. Numbers modified by Juhyoung Park.

#### Convolutional Operation

The convolutional operation consists of a series of dot products between the layer and the filter [1]. Each layer in a CNN is represented as a three-dimensional grid with height,

width, and depth, where the depth corresponds to the channels, such as the RGB channels in image data. The layer has a shape of $height \times width \times depth$, and each element in the layer is referred to as a pixel [1].

Aggarwal [1] represents a layer as $L_q \times B_q \times d_q$ where $L_q$ is the height, $B_q$ is the width, and $d_q$ is the depth. The filter has the same depth as the layer but is spatially much smaller, square, and typically of odd size. The number of filters in each layer is typically a power of two, and each filter detects a specific pattern or feature. It is represented as $F_q \times F_q \times d_q$ [1]. The convolutional operation can be described as follows:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)} \quad \begin{array}{l} \forall i \in \{1,\ldots,L_q - F_q + 1\}, \\ \forall j \in \{1,\ldots,B_q - F_q + 1\}, \text{ and} \\ \forall p \in \{1,\ldots,d_{q+1}\}, \end{array} \tag{2.10}$$

where $w^{(p,q)}$ denotes the $p$th filter in the $q$th layer, and $h^{(q)}$ denotes the $q$th layer. The result $h^{(q+1)}$ corresponds to the $(q+1)$th layer. $i, j, k$ denote the height, width, and depth indices of the layer, and $r, s, k$ denote the height, width, and depth indices of the filter [1].

This operation can be simply explained as follows: the filter is moved from left to right and from top to bottom over the input; at each location, it performs a pixel-wise dot product with the corresponding region and sums the resulting products, thereby reducing the spatial size of the output. Padding can be applied before the operation to retain the same size as the input [1].

**ReLU Activation**

The ReLU activation function, which is expressed in Equation 2.5, is commonly used in CNNs because it is computationally more efficient [1]. Krizhevsky et al. [31] demonstrated that a model using ReLU can be trained faster than a model using the tanh activation

function, even when both achieve similar performance. Since it is applied independently to each pixel in the layer, the output retains the same spatial dimensions as the input [1].

**Pooling Operation**

The pooling operation is similar to the convolutional operation; however, since it is applied to each depth channel separately, the output has the same number of depths as the input [1]. Most commonly, max-pooling is used, which returns the maximum value among the pixels within the pooling window. In contrast, other poolings, such as average pooling and weighted average pooling, are used much less frequently. The pooling operation has several benefits, one of which is that it reduces the size of the data while preserving the important features of the input [1, 19].

### 2.1.5 Evaluation Metrics Used for Neural Networks

There are several ways to evaluate the performance of neural networks, such as accuracy, precision, recall, and F1-score. In this study, we mainly used accuracy as the primary evaluation metric, while treating the other metrics as complementary reference measures. We also found that the `evaluate` method in TensorFlow produced slightly overestimated results. Our calculations performed manually yielded lower accuracy, which matched the results obtained using methods from the Scikit-learn library. Therefore, for the remainder of our experiments, we adopted the evaluation methods provided by the Scikit-learn library, using more detailed metrics such as Precision, Recall, and F1-score.

**Accuracy**

Accuracy measures the number of correctly predicted instances among all instances [22]. It is defined as:

$$Accuracy = \frac{TP + TN}{P + N}, \tag{2.11}$$

where $TP$ denotes true positives, which are actual positive instances correctly labelled by the model, and $TN$ denotes true negatives, which are actual negative instances correctly

labelled by the model. $P$ denotes the total number of actual positive instances, and $N$ denotes the total number of actual negative instances. Thus, $P+N$ corresponds to the total number of instances [22].

### Precision

Precision measures the number of correctly predicted positive instances among all instances predicted as positive [39]. It is defined as:

$$Precision = \frac{TP}{TP+FP}, \tag{2.12}$$

where $FP$ denotes false positives, which are actual negative instances incorrectly labelled as positive by the model [22, 39].

### Recall

Recall measures the number of correctly predicted positive instances among all actual positive instances [39]. It is defined as:

$$Recall = \frac{TP}{TP+FN}, \tag{2.13}$$

where $FN$ denotes false negatives, which are actual positive instances misclassified by the model. Recall is also known as sensitivity or the true positive rate [22, 39].

### F1-score

The F1-score, which is the harmonic mean of precision and recall, is a useful metric for measuring model performance when the dataset is class-imbalanced [39]. It is defined as:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{2.14}$$

This measure is more strict, as achieving a high score requires both precision and recall to be high [39]. However, it does not consider true negatives, which are actual negative instances that were correctly labelled as negative by the model, so it should be interpreted with care and complemented by other metrics, such as ROC-based measures, Informedness, or MCC [44].

## 2.2 Audio Features for Musical Instrument Classification

Deep neural networks introduced in the previous section are widely used in many fields of research, and CNNs have become more and more popular after demonstrating superior performance in computer vision. In order to utilize these networks, the datasets are usually preprocessed into the appropriate format for each neural network. The same applies to datasets for MIR research because audio signals themselves are not suitable to feed into the neural networks [10].

Audio features are extracted from the audio signal, such as speech and music, which play a central role in MIR research [9]. Although audio signals themselves can be considered as information, more specific and dense information must be obtained through computational processing, yielding feature representations that are better suited to the objective of each task [9, 34]. This section introduces the audio features employed in this study.

### 2.2.1 Mel-spectrogram and Mel-frequency Cepstral Coefficients

Mel-spectrogram and Mel-frequency Cepstral Coefficients (MFCCs) are the most popular audio features in MIR research, and to obtain both features, the Short-time Fourier Transform (STFT) is needed first [34].

#### Short-time Fourier Transform

The most fundamental tool for extracting audio features is the short-time Fourier transform (STFT) [34]. It is usually applied in the first stage of many feature extractions to obtain the distribution of signal power over time and frequency. It can be interpreted as

performing a discrete Fourier transform (DFT) on short, windowed segments of the signal. The DFT yields a single spectrum for the entire signal, and the STFT yields a sequence of spectrums, one for each windowed segment of the signal [34]. Given a signal $x[n]$, the STFT is defined as:

$$X_{\hat{n}}(e^{j\omega}) = \sum_{m=-\infty}^{\infty} w[\hat{n}-m]x[m]e^{-j\omega m}, \tag{2.15}$$

where $\hat{n}$ denotes a time index and $w[\hat{n}-m]$ denotes a window sequence that emphasizes the portion of the input signal around time $\hat{n}$. The variable $\omega$ is the frequency variable, with $0 \leq \omega < 2\pi$ [45]. The complex exponential $e^{j\omega}$ is defined by Euler's formula: $e^{j\omega} = \cos\omega + j\sin\omega$. The term $e^{-j\omega m}$ is used as a complex exponential basis function at frequency $\omega$, used to extract the component of the signal at that frequency [34].

**Mel-spectrogram**

The mel-spectrogram is commonly used as input for CNNs in MIR research [10]. The audio signal is first transformed into the frequency domain using the STFT, yielding $X_m[k]$, the spectrum corresponding to the $m^{th}$ time frame. The magnitude spectrum $X_m[k]$ is then passed through a Mel-scale filterbank, implemented as a set of triangular filters applied to the STFT frequency bins [45].

The mel-scaled spectrum of the $m^{th}$ frame is defined as:

$$MF_m[r] = \frac{1}{A_r}\sum_{k=L_r}^{U_r} |V_r[k]X_m[k]|^2, \quad r = 1,\ldots,R, \tag{2.16}$$

where $R$ is the total number of filters, $V_r[K]$ is the weight function for the $r^{th}$ filter over the index range $[L_r, U_r]$, and $A_r$ denotes a normalization factor for the $r^{th}$ mel filter, given by:

$$A_r = \sum_{k=L_r}^{U_r} |V_r[k]|^2. \tag{2.17}$$

This normalization is needed to produce a flat Mel-spectrum from a flat input Fourier spectrum [45]. The resulting spectrogram is usually rescaled by applying a logarithmic

transformation to its magnitude before being used as input to CNNs, which leads to a significant improvement in performance. This representation is referred to as the log-mel spectrogram [11].

**Mel-frequency Cepstral Coefficients**

Mel-frequency cepstral coefficients (MFCCs) are one of the most widely used audio features as input to ANNs [34]. MFCCs are obtained from the Mel-scaled spectrum as:

$$mfcc_m[n] = \frac{1}{R}\sum_{r=1}^{R} \log(MF_m[r]) \cos\left[\frac{2\pi}{R}\left(r+\frac{1}{2}\right)n\right], \tag{2.18}$$

where, for each frame $m$, a discrete cosine transform is applied to the logarithm of the Mel-filterbank outputs to obtain $mfcc_m[n]$ [45]. This processing reduces the dimensionality of the data while retaining the informative components of the signal [34].

### 2.2.2 Constant-Q Transform

The Constant-Q Transform (CQT) is another time–frequency representation, originally proposed by Brown [8]. The STFT is useful for analyzing the audio frequencies. But it does not represent the frequency in a manner that aligns with the human musical perspective, since it uses the same resolution for each frequency bin. In contrast, the CQT parameter uses frequency bins spaced at 1/24th of an octave, maintaining a constant ratio between frequency and resolution across all bands. Due to its constant-Q design, the filterbank is particularly well-suited to represent musical notes [8, 53].

Given a signal $x[n]$ at the time index $n$, the CQT is defined as:

$$X^{CQ}(k,n) = \sum_{j=n-\lfloor N_k/2 \rfloor}^{n+\lfloor N_k/2 \rfloor} x(j) a_k^*(j-n+N_k/2), \quad k = 1,\ldots,K, \tag{2.19}$$

where $k$ denotes the index of the frequency bin and $K$ is the total number of bins [53]. The term $a_k^*(n)$ denotes the complex conjugate of $a_k(n)$, which is a complex-valued atom

defined as:

$$a_k(n) = \frac{1}{N_k} w\left(\frac{n}{N_k}\right) exp\left[-i2\pi n \frac{f_k}{f_s}\right], \tag{2.20}$$

where $f_k$ represents the centre frequency of bin $k$, $f_s$ is the sampling rate, and $w(t)$ is a window function sampled at time $t$. The window lengths $N_k$ are proportional to $f_k$, therefore ensuring the same Q-factor for all bins [53]. The centre frequency $f_k$ is defined as:

$$f_k = f_1 2^{\frac{k-1}{B}}, \tag{2.21}$$

where $f_1$ represents the center frequency of the lowest frequency of interest to be included in the transform, and $B$ is the number of bins per octave, which is a key parameter as it determines the resolution of the output. The window lengths $N_k$ is defined as:

$$N_k = \frac{q f_s}{f_k(2^{\frac{1}{B}} - 1)}, \tag{2.22}$$

where $0 < q \leq 1$ is a scaling factor, and $q$ is usually 1 [53].

### 2.2.3 Chroma Energy Normalized Statistics

Chroma Energy Normalized Statistics (CENS) are chroma-based features that present a high tolerance to variations in dynamics, timbre, and articulation [40]. Therefore, they are widely used in audio matching and retrieval applications, where robustness to performance-related variations is essential [32, 40]. They are also well-suited for capturing chord or harmonic progressions, making them particularly appropriate as features for Western music [40].

The first step of extracting CENS feature is converting the audio into 88 frequency bands, which correspond exactly to the pitches of the keys on a modern 88-note piano, ranging from A0 to C8 [40]. This can be achieved either by employing elliptic filters or by using the same filters as those used in the CQT [32, 40].

Subsequently, for each of the 88 frequency bands, the short-time mean-square power

(STMSP) is obtained by computing the mean-square value of the signal in that band using a 200-ms rectangular window with 50% overlap. The 88 STMSP values are then aggregated into 12 chroma classes by summing the STMSPs of all pitches belonging to the same chroma (e.g., A0–A7 for chroma A).

Finally, for each window, the energy distribution over the 12 chroma classes is obtained by normalizing over the 12 chroma values [40].

### 2.2.4 Zero-crossing Rate

The zero-crossing rate (ZCR) is defined as the number of times a signal changes sign and crosses the zero level within a given time window or frame, divided by the duration of that window [15]. It is useful for separating voiced and unvoiced parts of speech and is also effective for distinguishing percussive sounds and non-percussive sounds, which are important characteristics of many musical instruments [15, 20].

The ZCR is expressed as:

$$Z_n = \sum_{m=-\infty}^{\infty} |sgn[x(m)] - sgn[x(m-1)]| w(n-m), \tag{2.23}$$

where $sgn[x(n)] = 1$ when $x(n) \geq 0$ and $sgn[x(n)] = -1$ when $x(n) < 0$, and $w(n)$ is a window function of size $N$, defined as $w(n) = 1/2N$ for $0 \leq n < N-1$ and $w(n) = 0$ elsewhere [15].

## 2.3 Principal Component Analysis

An efficiently represented dataset of appropriate size can be highly beneficial in a variety of applications [19]. We use longer lengths of audio clips than in previous work, which will be discussed in Section 2.4, producing a much larger size of the entity as input data. We employ the Principal Component Analysis (PCA) for dimensionality reduction to reduce the size of the data while preserving the most informative components.

### 2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the oldest techniques for statistical data analysis. It used to be difficult to apply in practice because the calculations were hard to perform by hand. But it has become widely used with the advent of modern high-performance computing environments [29].

PCA processing first derives a linear combination of the original variables that maximizes the variance of the data, defining this as the first principal component. It then successively extracts subsequent components that maximize the remaining variance, under the constraint that each new component is uncorrelated with all previously derived components [1, 29].

PCA can be obtained by an eigendecomposition of the covariance matrix [55]. Let $P$ be an orthonormal matrix such that, $Y = PX$, where $X$ is dataset, and covariance matrix $C_Y = \frac{1}{n}YY^T$ is diagonal, where $n$ is the number of samples. The rows of $P$ correspond to the principal components of $X$ [55]. $C_Y$ can be written as follows:

$$C_Y = \frac{1}{n}YY^T = \frac{1}{n}(PX)(PX)^T = \frac{1}{n}PXX^TP^T = P(\frac{1}{n}XX^T)P^T = PC_XP^T, \tag{2.24}$$

where $C_X$ is the covariance matrix of $X$, which can be written as follows:

$$C_X = \frac{1}{n}XX^T = EDE^T, \tag{2.25}$$

where $E$ is a matrix of eigenvectors of $C_X$, and $D$ is a diagonal matrix. Substituting $C_X = EDE^T$ into Equation 2.24 and choosing $P = E^T$, we obtain the following:

$$C_Y = PC_XP^T = E^T(EDE^T)E = D. \tag{2.26}$$

As a result, the eigenvectors of $C_X$ are the principal components of $X$ [55].

This approach effectively reduces data dimensionality and complexity by retaining only

the most informative components. However, because each principal component is constructed as a linear combination of all the meaningful variables, individual components cannot be easily interpreted meaningfully [1, 29].

### 2.3.2 Incremental Principal Component Analysis

Incremental Principal Component Analysis (Incremental PCA) is an alternative approach for computing PCA to handle larger datasets [35]. Traditional PCA requires loading the entire dataset into memory, whereas Incremental PCA does it more efficiently by decomposing the computation into smaller matrix operations and updating the result as new data arrives. This technique is useful for handling growing datasets or streaming data [35].

Incremental PCA can be obtained by calculating $Q_{n+1}$ from $Q_n$, where $Q_n$ denotes the covariance matrix of the standardized data matrix $Z_n$ at time step $n$ [35]. Let $x_n$ be a row vector in step $n$, and $x_{n+1}$ be a new observation. The standardized data matrix $Z_{n+1}$ can be written as follows:

$$Z_{n+1} = \begin{bmatrix} Z_n\Sigma_n + \Delta \\ y \end{bmatrix} \Sigma_{n+1}^{-1}, \tag{2.27}$$

where $\Sigma_n$ and $\Sigma_{n+1}$ are diagonal matrices of sample standard deviations at steps $n$ and $n+1$, respectively, $\Delta$ is obtained by repeating the row vector $\delta = \bar{x}_n - \bar{x}_{n+1}$ $n$ times, where $\bar{x}_n$ and $\bar{x}_{n+1}$ are the sample means at steps $n$ and $n+1$, respectively, and $y = x_{n+1} - \bar{x}_{n+1}$. Then, according to Lippi and Ceccarelli [35], the covariance matrix can be updated as

$$nQ_{n+1} = \Sigma_{n+1}^{-1}\Sigma_n Q_n \Sigma_n \Sigma_{n+1}^{-1} + n\Sigma_{n+1}^{-1}\delta^\top \delta \Sigma_{n+1}^{-1} + z^\top z, \tag{2.28}$$

where $\delta = \bar{x}_n - \bar{x}_{n+1}$ and $z = y\Sigma_{n+1}^{-1}$.

However, because exact incremental updates can introduce discontinuities, such as arbitrary eigenvector sign flipping and component reordering due to variance shifts, applying a continuity correction algorithm is essential to maintain consistent principal components [35].

## 2.4 Related Works in Musical Instrument Classification

A history of musical instrument classification research can be broadly divided into two eras: before and after CNNs became the dominant approach. Before researchers recognized the potential of CNNs, most studies focused on proposing new audio feature extraction methods with statistical and traditional machine learning techniques. ANNs were also used early, but they achieved only limited performance [25].

Most studies also focused on solo or monophonic instrument classification. However, following the successive performance of AlexNet, research on multi-class, multi-label musical instrument classification using CNNs has grown more popular, which has become the dominant approach, as traditional methods were not well-suited to this [10].

### 2.4.1 Statistical and Traditional Machine Learning Approaches

Choosing appropriate audio features or proposing new audio feature extraction methods is very important in musical instrument classification research. Although this remains an important topic using CNNs, it is arguably even more critical for statistical and traditional machine learning approaches.

#### Feature-Based Instrument Classification using Traditional Methods

Simmermacher et al. [56] investigate methods for classifying classical instruments by comparing various feature sets and classifiers. They employ the perception-based features, MPEG-7-based features, and MFCCs. As for the classifiers, they employ condensed k-nearest neighbours (k-NN), multilayer perceptron (MLP), and Support Vector Machine (SVM). Their experiments are classifying 20 instruments from the University of Iowa Electronic Music Studios samples (UIOWA IMS) and classifying solo instruments from CD recordings. They find that MFCCs are the most crucial features among those they employ, and that adding a small set of empirically selected features from MPEG-7 descriptors yields a modest additional improvement in classification performance. They also find that the MLP classifier consistently outperforms the other classifiers [56]. Their study is limited

to solo classical instruments. But their recognition of the potential of MLPs appears to be a key outcome of their experiments.

**Pairwise Statistical Classification Using Support Vector Machine**

Essid et al. [16] address the task of classifying instruments from different instrument families by pairwise classification strategies. Their method classifies 10 solo classical instruments from CDs. They employ the new feature called Octave Band Signal Intensities (OBSI) and their ratio (OBSIR), which utilizes the power distribution of the harmonics of the sound. They also introduce feature selection algorithms, such as IRMFSP, GAFS, and Class Pairwise Feature Selection. They employ the Gaussian Mixture Model (GMM) with SVM for classification, which shows the best result with Radial Basis Function (RBF) kernels with longer audio data [16].

**Likelihood–Frequency–Time Analysis with Support Vector Machine**

Özbek et al. [42] address musical note and instrument classification using SVM and a likelihood–frequency–time (LiFT) analysis. The LiFT utilizes the constant-Q filterbank to compute the likelihoods of the output being only noise or both signal and noise. They use the UIOWA IMS dataset of 19 classical instruments. They find that the SVM achieves higher accuracy when the number of classes is smaller, and that grouping instruments depends on their articulation characteristics. They also find that the LiFT approach is more suitable for pitch detection than for instrument classification because the filterbank effectively captures harmonic partials [42]. Their finding on grouping instruments by articulation is particularly insightful.

**A Hierarchical Method using K-Nearest Neighbours**

Jiang et al. [28] propose a cascade classifier for polyphonic multi-label musical instrument recognition, a hierarchical method built on clustered audio features that captures not only inter-instrument similarity but also differences in playing techniques. They use only

the left channel of audio from the McGill University Master Samples (MUMS) library, containing single notes of 45 instruments with different pitches and various playing techniques for training. Their test set consists of mixtures of two instrument samples from the training set. They also experiment with mixtures of three instrument samples. They use a K-Nearest Neighbour (KNN) classifier with $k = 3$, employing the spectral flatness coefficient for instrument family classification and the power spectrum for bottom-level classification [28]. Their approach is well-motivated for polyphonic multi-label classification.

**Statistical Classification using Individual Partials**

Barbedo and Tzanetakis [3] address the problem of classifying musical instruments in polyphonic musical data. They propose a method that classifies 25 instruments from the RWC database. They extract each note and then generate samples that have 2 to 5 instruments and 0.5 to 5 seconds of duration. Their method first splits the audio into partials, then filters them to suppress other frequencies that belong to different instruments, and then applies the pairwise method. After extracting 34 features from the partials, 9 features are chosen for each pair. It first chooses a partial winner that has the highest number of votes. After performing this process for all pairs, the winner for a frame is chosen from the most voted partial winner. This process is performed in all frames for a winner for each sample. If the length of the sample is longer than 5% of the total length, it is classified as one of the instruments of the sample [3].

### 2.4.2 Deep Learning Approaches

From the mid-2010s, CNNs led musical instrument classification research to the next level. New audio feature extraction methods have continued to be proposed. But designing alternative CNN-based architectures has also become a popular direction. Multi-label classification has become more prevalent, and multi-class, multi-label classification has also appeared more frequently than before.

**Multiresolution Recurrence Plots using CNNs**

Park and Lee [43] propose a new feature, Multiresolution Recurrence Plots (MRPs), which has the phase information of audio data. They propose combining Mel-spectrograms and MFCCs with MRPs to achieve higher performance. They employ different block sizes and resolutions for constructing MRPs, which have seven layers. They then retain only the first MRP for each layer and apply max pooling to reduce the image size. They employ a square-root function to reduce the dynamic range of the MRPs, and then perform data augmentation. Their model classifies 20 instruments from the UIOWA MIS dataset and their own four piano virtual instruments, and the model trained using both spectrograms and MRPs achieves the best performance [43].

**Fractional Fourier Transform and the Counter Propagation Neural Network**

Bhalke et al. [4] propose a new feature called Fractional Fourier Transform (FrFT) based MFCCs and the Counter Propagation Neural Network (CPNN) for solo instruments. They employ eight audio features, including MFCCs, Cepstrum analysis, and FrFT-based MFCCs. FrFT is a variation of the Fourier Transform that emphasizes similarities within the same instrument family while revealing distinctions between different instrument families. The CPNN classifies 19 classical instruments among four musical instrument families from the MUMS database, which has Kohonen's Self Organizing Map (SOM) layer between the input and output layers. One of their experiments, conducted by adding Additive White Gaussian Noise (AWGN) to the audio signals, shows that FrFT-based MFCC features perform better than the original MFCC features [4].

**Convolutional Neural Networks Inspired by AlexNet and VGGNet**

Han et al. [23] propose a method to recognize the predominant instrument in polyphonic music using CNNs. It averages the two channels of a stereo audio signal to obtain a mono signal, then reduces the sample rate from 44,100 Hz to 22,050 Hz to remove frequencies above 11,025 Hz. It then normalizes the processed audio signals and converts them into

Mel-spectrogram images. Their model is influenced by the AlexNet and VGGNet. They use the IRMAS dataset and extract 11 pitched instruments from it, excluding other instruments such as percussion and bass. They use one-second audio segments for training, while test audio ranges from 5 to 20 seconds and is handled using a sliding window. The predominant instruments are then determined by aggregating the predictions over time and selecting those whose scores exceed a given threshold [23].

The use of a sliding window in their work for testing is effective and allows them to handle longer audio excerpts, but a key limitation is that merging a stereo audio signal into mono removes important spatial information and can introduce phase shifts, phase cancellation, and loss of panorama detail. Cutting off frequencies above 11,025 Hz may also be inappropriate. The frequency range that humans can hear is from 20 to 20,000 Hz, and the frequency range above 10 kHz presents the brightness of musical instruments and the airiness of the space [27].

**Attention Mechanism with ANNs**

Gururani et al. [21] propose using an attention mechanism for multi-label instrument classification. They use the OpenMIC dataset, which consists of 10-second audio clips annotated with 20 instrument classes. Each audio clip is converted into a Mel-spectrogram and split into short frames. These frames are passed through a pre-trained CNN provided with the OpenMIC dataset, and then passed through a fully connected network. The output is then fed into an attention layer, which computes attention weights for each instrument to aggregate frame-level predictions into its presence probability. They show that their attention model outperforms ANN, RNN, and Random Forest baselines [21]. The limitation of the study is that it relies on a weakly labelled dataset with the pre-trained CNN, which makes it difficult to obtain reliable frame-level labels.

**MIC_FuzzyNet: A Transfer Learning–Based Ensemble Model using CNNs**

The research by Sahoo et al. [51] addresses the problem of classifying musical instruments using CNNs. They propose the MIC_FuzzyNet model, which uses CQT, Mel-spectrogram, and Semitone spectrogram as inputs, forming a 3D matrix by stacking these three images, and employs an ensemble method. It utilizes transfer learning by employing EfficientNetV2 [58] and ResNet18 [24]. Their outputs are then used in the proposed fuzzy ranking method, which is based on the Gompertz function. They use 7 Persian monophonic instruments from the PCMIR dataset and 5 polyphonic instruments from the IRMAS dataset. The ensemble model using the fuzzy ranking method achieves the best result, and experiments with the PCMIR dataset yield higher accuracies due to its monophonic data [51]. The models they employ for transfer learning are not originally designed for music spectrograms, so their suitability may require further validation.

**CNN-based Model with Audio Data Augmentation for Percussive Instruments**

Mahanta et al. [36] propose a CNN-based method to classify 20 classical instrument classes from the Philharmonia and UIOWA MIS datasets. They employ data augmentation methods to address the class imbalance issue, using the Audiomentations Python library, applied only to the percussive instruments. The methods they employ are TimeStretch, PitchShift, Shift, Trim, and Gain. MFCCs are extracted from each 3.5-second audio excerpt and used as input to the CNN. They also employ a Stratified K-fold cross-validation with 10 folds to compare results between ANN and CNN [36].

The limitations of their study are that the dataset focuses on classical instruments, which may not be ideal for a real-world situation, and that the model is designed to classify only single instrument classes rather than multi-label classes. Nonetheless, techniques such as data augmentation and combining two datasets are valuable aspects of their approach, although applying TimeStretch and PitchShift to the dataset may not be ideal because they can produce unrealistic sounds.

### Spiking Neural Networks and Fully Modular Convolutional Neural Network

Blaszke et al. [7] propose a less complex neural network, Spiking Neural Networks (SNNs), alongside a modular model, a Fully Modular Convolutional Neural Network (FMCNN), to tackle multi-class, multi-label musical instrument classification. SNNs offer significant advantages for tasks involving temporal information and event-based computation, achieving strong performance relative to their model size. The FMCNN consists of modules. Each module is responsible for classifying a specific musical instrument. They classify 13 instruments, and the dataset is constructed by combining several existing audio datasets, and each audio sample is a mixture of a random number of instruments with randomly selected audio samples. For the SNN model, the MFCCs are fed into the network as one-dimensional convolutional inputs. For the FMCNN, the Mel-cepstrogram is extracted and fed into the network [7]. This research is well-structured, and it would be preferable to report the instrument distribution for each mixed sample. Their instrument selection also seems to be biased toward classical music.

### Compact Convolutional Transformers

Lekshmi and Rajan [48] propose Compact Convolutional Transformers (CCTs) to tackle the problem of multiple predominant instrument classification. Their model takes Mel-spectrograms. A CCT tokenizer composed of convolutional layers converts these spectrograms into tokens, and a positional embedding layer is used to encode positional information. These tokens, together with the positional information, are then fed into the transformer layers. It classifies 11 instruments from the IRMAS dataset. They train the model using 3-second audio excerpts, whereas for evaluation, they use audio samples ranging from 5 to 20 seconds in length. They demonstrate that their CCT-based model achieves better results than the model proposed by Han et al. [48].

**Instrument Classification with Convolutional KAN**

The research by Zheng et al. [63] proposes the Instrument Classification with Convolutional KAN (ICKAN) model. The Kolmogorov–Arnold Network (KAN) is a neural architecture proposed as an alternative to multilayer perceptrons (MLPs). In ICKAN, a CNN is used to uncover latent patterns in the audio features, while the KAN makes timbral characteristics more salient through learnable nonlinear mappings. Their dataset has 20 instruments, including keyboard, string, woodwind, brass, and percussion families, and each 10-second audio clip are rendered from MIDI files using virtual instruments software. Their experiments are conducted using both Mel-spectrograms and MFCCs; while the Mel-spectrogram yields the best performance, the performance obtained with MFCCs is also competitive [63].

Employing KAN is impressive because of its ability to detect the timbral characteristics of musical instruments, but constructing a dataset by exporting audio from MIDI files using virtual instrument software may not be ideal, as both the MIDI files and the virtual instruments have limited ability to capture the nuances of human performance and the authentic characteristics of real instruments. Moreover, if only a few virtual instruments are employed, the timbral palette would be very limited.

**Hierarchical Deep Learning for Minority Instrument Detection using CNNs**

Sechet et al. [54] propose a hierarchical deep learning approach for classifying minority instruments. They use 94 tracks out of 122 pieces from the MedleyDB dataset and segment each track into non-overlapping one-second audio excerpts. The instruments are grouped into five categories based on their sound production mechanism. They use a CNN inspired by VGGNet that takes MFCCs as input and applies sigmoid activations to produce an 85-dimensional binary output vector indicating the presence or absence of each instrument. The model follows a hierarchical strategy, where it first predicts the eight instrument groups, and in a subsequent stage, classifies the individual instruments within the predicted group

[54]. Their choice of dataset seems very reasonable. However, the limited quantity of audio samples for minority instruments is one of the most critical issues that must be addressed in every MIR study.

## 2.5 Summary

This chapter provided background information to help the reader understand the rest of our research. It first described the background knowledge of artificial neural networks and convolutional neural networks, followed by the evaluation metrics, including Precision, Recall, and F1-score.

It then outlined the fundamental background of the audio features employed in our experiments, namely the Mel-spectrogram, which provides a time–frequency representation of the audio energy distribution; Mel-frequency cepstral coefficients (MFCCs), which offer a dimensionally reduced representation of the Mel-frequency spectrum; the Constant-Q transform (CQT), which yields a frequency representation more closely aligned with music; Chroma Energy Normalized Statistics (CENS), which capture robust, tempo-insensitive harmonic patterns such as chord progressions; and the zero-crossing rate, which is informative about noisiness and the presence of transient, unvoiced, or percussive components.

It then described the Principal Component Analysis (PCA) and the Incremental PCA, which are used to reduce the dimensionality of the data while retaining the key information necessary for discrimination. Incremental PCA can handle much larger data than normal PCA can.

Finally, it introduced the related works, providing an overview of the development of research on musical instrument classification, from statistical and traditional machine learning methods to deep neural networks with their variants and associated feature extraction techniques.

# Chapter 3

# Datasets and Preprocessing

In this chapter, we will provide detailed information about the datasets we employed for musical instrument classification, followed by a description of the data preprocessing. In the preprocessing stage, we first excerpt 1-minute audio clips and then extract the audio features. Additional processing, such as upsampling and audio feature dimensionality reduction, is applied for efficient and effective experiments.

## 3.1 Datasets

In Music Information Retrieval (MIR), obtaining datasets that are suitable for each study is one of the most significant and challenging issues. The biggest challenge is copyright. Unless we record all the audio tracks ourselves, obtaining permission from copyright holders for each piece of music can be very difficult or even impossible.

The quantity and quality of the audio data are also critical, as the dataset size strongly affects the training of generalized neural networks. MIDI (Musical Instrument Digital Interface) data are useful, depending on the research topic, as they contain information about which instruments play which notes, similar to a band or orchestra score rather than audio signals. However, when synthesizers or virtual instruments are used to convert MIDI data into audio files, it may be difficult to represent the full variety of actual instruments. In addition, Many datasets for MIR are biased toward classical music, which makes it challenging to apply them to contemporary music in real-world settings. Metadata is also crucial. If the labelling is not done correctly, the trained model may not be reliable.

The datasets we selected are MUSDB18[1] and MedleyDB 2.0[2], which are largely free from these issues, except for limited data quantity. Both datasets provide individual instrument tracks for each piece of music. We first used the MUSDB18 to conduct pilot studies and then extended the experiments using the MedleyDB, moving from a smaller to a relatively larger dataset.

Each dataset provides its own Python package, which facilitates collecting instrument labels for each piece of music. However, this information is mostly provided as strings, so it must be converted to match the input format required by training models [17]. We use One-hot encoding and Label encoding.

### One-hot Encoding

One-hot encoding represents the presence of a class as a binary number. For example, if a guitar is present in an audio clip, its value is 1, and if it is absent, its value is 0 [17]. Since our models perform multi-class, multi-label instrument classification, each audio clip is associated with a binary vector as its label. An example illustrating the presence of drums and bass in an audio clip from the MUSDB18 dataset is shown in Table 3.1.

Table 3.1: Example of one-hot encoding indicating the presence of drums and/or bass in an audio clip.

| Instrument | Label |
|---|---|
| Drums | [1, 0] |
| Bass | [0, 1] |
| Drums & Bass | [1, 1] |

### Label Encoding

One-hot encoding provides a simple way to represent the presence of multiple instruments. However, it may not be optimal because it can encode more label combinations than actually occur in the data. For example, a binary vector of length 2 can represent $2^2$ label

[1]MUSDB18, `https://sigsep.github.io/datasets/musdb.html/`.
[2]MedleyDB official website, `https://medleydb.weebly.com/`.

combinations, from $[0,0]$ to $[1,1]$. However, if the dataset does not contain any samples with the label $[0,0]$, which corresponds to a silent audio clip or an audio clip with no drums and bass, this can result in misclassification. Due to this potential misclassification, this method does not always achieve better performance than other encoding approaches. It typically shows competitive performance [2]. Therefore, we remap each label to the index as shown in Table 3.2 when using the Softmax activation function in our models.

Table 3.2: Example of labelling for the audio in Table 3.1.

| Category | Index |
|---|---|
| drums | 0 |
| bass | 1 |
| drums_bass | 2 |

### 3.1.1 MUSDB18

The MUSDB18 dataset [46] was created by SigSep[3] in 2017 to support research on audio source separation. It consists of 150 pieces of music in various genres, including Pop, Jazz, Rock, Country, and Electronic. 100 tracks of them are taken from the DSD100 dataset[4], which is derived from the "Mixing Secrets" Free Multitrack Download Library[5], 46 tracks are taken from MedleyDB, 2 tracks are provided by Native Instruments, and the remaining 2 tracks are contributed by the band "The Easton Ellises".

Each piece of music is stored in the Native Instruments STEMS format [41], which was developed by Native Instruments GmbH. This format was designed to be used in live DJ performance with the company's Traktor DJ software. Although the file extension of the STEMS format is ".mp4", each file contains four lossy-compressed stereo stem tracks for each class, typically corresponding to drums, bass, an instrument for a melody part, and other instruments [41]. In the MUSDB18 dataset, these are provided as four stereo tracks representing drums, bass, vocals, and other instruments. Using the accompanying

[3]SigSep, `https://sigsep.github.io/`.
[4]DSD100, `https://sigsep.github.io/datasets/dsd100.html`.
[5]"Mixing Secrets" Free Multitrack Download Library, `https://www.cambridge-mt.com/ms3/mtk/`.

Python package[6], one can derive both the full mixture of these four individual tracks and an instrumental mixture, which is the mixture without vocals [46].

Each piece of music is decoded as a 16-bit, 44,100 Hz stereo WAV file using the MUSDB18 Python package. After this step, seven (7) out of the 150 pieces are excluded due to tagging errors [46]. Consequently, a total of 143 pieces of music are used in our experiments, each providing four stems. Additional processing will be discussed in Chapter 4.

### 3.1.2 MedleyDB

MedleyDB [5, 6] was constructed in 2014 to facilitate research on melody extraction. The first release comprises 122 multitrack pieces in various genres, including classical, rock, jazz, and pop. Those are contributed by independent artists, NYU's Dolan Recording Studio, Weathervane Music, and Music Delta [5].

The dataset provides both raw mono tracks and stereo stem tracks. The stereo stems are mixtures of related mono sources. For example, left and right piano microphones are mixed into a stereo piano stem. Each piece is accompanied by a YAML file containing metadata, such as the artist name, title, genre, and instrument labels for each stem and raw track. In addition, annotations, including instrument activations and melody annotations, are provided in separate files in .csv format [5].

Its extended version, MedleyDB 2.0 [6], was released in 2016. It included 74 additional multitrack recordings in the same format. Their Python tools for accessing the dataset and its annotations are provided on GitHub[7] [6].

According to the MedleyDB 2.0 report, the dataset contains 254 multitracks after the addition of 132 tracks [6]. However, the official website documents 74 additional tracks for a total of 196 multitracks. We use the track list and counts from the official MedleyDB website in our study.

[6] A Python package for the MUSDB18 dataset, `https://github.com/sigsep/sigsep-mus-db/`.
[7] Annotation and Python tools for MedleyDB, `https://github.com/marl/medleydb/`.

The number of distinct instruments is 122 at the raw track level, which is reduced to 84 when tracks are summed into stems. However, 84 instrument labels are still too many to be recognized, even by professional musicians or music producers. Therefore, we grouped related instruments into a smaller set of instrument classes, such that the total number of classes does not exceed 10, where the classification task becomes more feasible. The details of this labelling scheme will be described in Section 4.3.1.

## 3.2 Preprocessing

Our preprocessing pipeline consists of two stages: audio preprocessing and audio feature extraction. In the audio preprocessing stage, we removed quiet background noise or silent segments and merged the remaining audio into a single clip. We then generated summed audio clips for multiple instruments. In the audio feature extraction stage, we extracted audio features using the librosa[8] Python library.

### 3.2.1 librosa Python Library

The librosa library [37] is one of the most widely used libraries in MIR research. It also plays a central role in handling audio data and extracting audio features in our study.

Historically, researchers often relied on MATLAB or C++ for audio and signal processing experiments. However, with the rise of machine learning and the growth of the Python ecosystem, libraries such as NumPy[9] and Scikit-learn[10] have made Python an increasingly popular choice for research. Consequently, librosa was developed as a Python library so that people can more easily apply MIR techniques and machine learning methods within a unified environment [37]. It represents audio data as a one-dimensional NumPy array, and the parameters of its methods are standardized, resulting in a low learning curve for MATLAB users. It supports basic file input/output and signal processing operations, in-

[8] librosa, python library for music and audio analysis, `https://librosa.org/`.
[9] NumPy, for scientific computing, `https://numpy.org/`.
[10] Scikit-learn, an open source machine learning library, `https://scikit-learn.org/`.

cluding time-domain processing, pitch tracking, and audio feature extraction such as Mel and chroma features. Its analysis and decomposition functions, such as beat tracking and harmonic/percussive separation, are also useful. In addition, it can visualize audio data by displaying spectrograms [37].

### 3.2.2 Audio Preprocessing

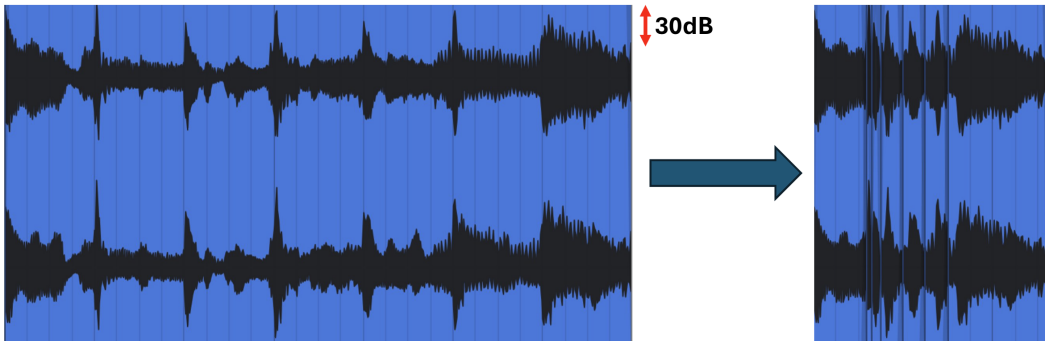
In our work, audio preprocessing consists of two main steps: excerpting and summing audio clips. After each audio file was loaded, it was first saved in the corresponding instrument folder during the excerpting step. Subsequently, to obtain a single audio clip from multiple instruments, we performed a summation over the instrument tracks we want to include.

We excerpted an 1-minute length of audio clip from an offset position in each audio file to preserve diverse musical phrase information because a musical instrument is characterized not only by its timbre but also by its playing style or articulation. In addition, since each channel in a stereo audio file carries distinct information that represents the spatial distribution of instruments within the stereo field [18], we retain the original stereo (left and right) format.

#### Excerpting Audio Clip

We removed silent regions from each audio clip. A threshold of 30 dB is used to check silence, meaning that regions whose level is 30 dB below the maximum are treated as silent. Figure 3.1 represents this procedure.

This threshold parameter was chosen empirically by listening and comparing multiple candidate values, which provides a good compromise between removing genuinely silent regions and preserving low-level but meaningful signals. In addition, it offers a certain degree of robustness against leakage from other instruments (bleeding) in multitrack recordings. For example, when vocal leakage occurs on a drum track during recording, the proposed processing can remove the leaked vocal component from the drum track. This




(a) Before trimming (b) After trimming

Figure 3.1: Example of waveform changes in the excerpting step.

procedure is implemented using `librosa.effects.trim` and `librosa.effects.split` methods.

It is possible that some meaningful parts are removed by this processing. But we assumed that only unwanted regions would be discarded. In practice, our informal listening tests indicated that the results were satisfactory for the experiments and did not sound overly artificial.

After the trimming process, we retained a 2-minute clip. If the duration was shorter than 2 minutes, the clip was duplicated and concatenated with the original signal until its length exceeded 2 minutes. We then selected an 1-minute region starting from a specified offset. The offset value can vary. Since the chorus part, just before the outro, typically contains the main musical phrase that we aim to preserve, a region in the second half was generally preferable.

The final output clips for each instrument were then saved in the corresponding instrument folder with their own index numbers.

**Summing audio clips**

We generated audio clips with various combinations of instruments. Although many summation strategies are possible, we adopted the simplest approach. Randomly selected audio clips from the chosen instrument folders were loaded, and then the audio clips were summed and averaged. According to Sturmel et al. [57], the summing process is defined

for each sample $n$ as:

$$m_j(n) = \sum_i a_{i,j} s_i(n), \tag{3.1}$$

where $m_j$ represents the mixture signal in the output channel $j$, $s_i$ denotes the $i$th input source signal, and $a_{i,j}$ is the mixing weight of the source $i$ in the output channel $j$ [57]. For our study, this equation can be rewritten as:

$$m(n) = \frac{1}{N} \sum_{i=1}^{N} s_i(n), \tag{3.2}$$

where $N$ is the number of input audio clips.

### Upsampling

We used upsampling (oversampling) techniques to increase the number of audio clips for experiments on the MedleyDB dataset. Upsampling is applied to imbalanced datasets to make the same number of samples in each class by modifying or synthesizing data. It does not discard any existing data. Upsampling can lead to overfitting, and synthesized data can introduce noise into the model [1, 38].

SMOTE, Borderline SMOTE, and ADASYN [38] are widely used upsampling techniques, and data transformations or augmentations are also effective options. In our study, we performed upsampling by applying different normalization methods and adding Gaussian noise to the audio clips. The details of the experiments using upsampling techniques will be presented in Section 4.3.1.

### Adding Gaussian Noise

We chose to add Gaussian noise to the audio clips before the audio feature extraction step because other methods, such as TimeStretch and PitchShift, can produce unrealistic sounds. We used the Audiomentations[11] library. This library provides two methods to add

[11] Audiomentations, a Python library for audio data augmentation, `https://iver56.github.io/audiomentations/`.

Gaussian noise: `AddGaussianNoise` and `AddGaussianSNR`. The former adds noise with an absolute amplitude, whereas the latter adds noise with a relative amplitude determined by the Signal-to-Noise Ratio (SNR). We used the latter, with the SNR range set between 20 and 35 dB.

### 3.2.3 Audio Feature Extraction

The audio feature extraction step is slightly simpler than the previous step. Each audio clip was loaded and passed to the corresponding feature extraction methods. We used a frequency range from 30 Hz to 16 kHz. For bass instruments, content above 30 Hz is generally informative. Although frequencies above 16 kHz may contain some meaningful signal, we ignored them because they are increasingly difficult to perceive, especially for older listeners. We also used the default normalization method provided by each audio feature extraction method.

Each audio clip's extracted features for experiments with artificial neural networks (ANNs) has the shape $2 \times 137 \times 5168$, corresponding to 2 channels, 137 features, and 5168 frames. The 137 features consist of 40 from Mel-frequency Cepstral Coefficients (MFCCs), 84 from Constant-Q Transform (CQT), 12 from Chroma Energy Normalized Statistics (CENS), and 1 Zero-crossing rate value per frame. As for the experiments with convolutional neural networks (CNNs), the details will be provided in Chapter 5.

#### Mel-spectrogram

We used the `librosa.feature.melspectrogram` method, with 256 Mel filterbanks as the number of features. To convert the result into PNG image format, we applied a logarithmic transformation using the `power_to_db` method, followed by normalization and scaling by 255.

#### Mel-frequency Cepstral Coefficients

We used the `librosa.feature.mfcc` method with 40 MFCC features. For the Mel filterbanks parameter, we used 256. This method is implemented using the `melspectrogram` method, followed by the `power_to_db` method and the discrete cosine transform `fft.dct` method, as defined in Equation 2.18.

#### Constant-Q Transform

We used the `librosa.cqt` method with 84 bins, which corresponds to 7 octaves with 12 bins per octave. The lowest bin is centred at approximately C1 (32.70 Hz) [37]. The extracted feature from the CQT is a complex number. We used only the real part and discarded the imaginary part, which may not be optimal, but it reduces the feature size while still preserving meaningful information.

#### Chroma Energy Normalized Statistics

We used the `librosa.feature.chroma_cens` method with 12 chroma bins, 7 octaves, and 36 bins per octave. The CENS features can be computed from STFT-based or CQT-based chroma representations, and librosa uses the `chroma_cqt` method [37], which uses the recursive sub-sampling method [53].

#### Zero-crossing rate

We used the `librosa.feature.zero_crossing_rate` method with the center parameter set to False, and then matched its length to the other features using the `fix_length` method so that it could be stored together with other features in the same `numpy.ndarray`.

### 3.2.4 Audio Feature Dimensionality Reduction

Since we use longer lengths of audio clips than most previous works as discussed in Chapter 2, the resulting input size is $2 \times 137 \times 5168 = 1{,}416{,}032$ and becomes too large to build deeper neural networks. To address this issue, we employed the Principal Component

Analysis (PCA) for dimensionality reduction. We used the Scikit-learn library to implement both PCA and Incremental PCA. PCA was applied to the MUSDB18 dataset, while IncrementalPCA was used for MedleyDB. The details of the experiments using PCA will be presented in Chapter 4.

## 3.3 Summary

This chapter first described the datasets we used in our study. The MUSDB18 and MedleyDB datasets contain a wide variety of instruments with well-labelled metadata. Table 3.3 summarizes the datasets used in the experiments. The chapter then detailed the preprocessing stage, which includes audio preprocessing and audio feature extraction.

Table 3.3: Summary of datasets used in our experiments.

| **Dataset** | **# of tracks** | **# of Instrument stems** |
|---|---|---|
| *MUSDB*18 | 150 | 4 |
| *MedleyDB* | 198 | 84 |

The features extracted for the experiments using ANNs in Chapter 4 have the shape of $2 \times 137 \times 5168$, corresponding to 2 channels, 137 features (40 from MFCCs, 84 from CQT, 12 from CENS, and 1 zero-crossing rate), and 5168 frames. Additionally, we applied PCA and Incremental PCA to reduce the dimensionality of the extracted features, as their size is too large for ANNs. The additional features for the experiments using CNNs will be introduced in Chapter 5.

Lastly, we briefly introduced the data augmentation methods we employed. We used a couple of normalization methods and added Gaussian noise to the audio clips before extracting audio features. Further processing details will be discussed in Chapter 4 and Chapter 5.

# Chapter 4

# Classifying Musical Instruments using Artificial Neural Networks

This chapter will present empirical experiments on simple and hierarchical models employing Artificial Neural Networks (ANNs) using two datasets, the MUSDB18 and the MedleyDB. It will first introduce our proposed ANN architecture, followed by the empirical experiments on the two datasets.

The experiments were conducted in two stages for each dataset. In the first stage, a single neural network was used to examine its ability to classify a multi-class, multi-label dataset. In the second stage, hierarchical models consisting of multiple ANNs were constructed. The objective is to determine whether these hierarchical models outperform the single neural network.

We first conducted ANNs using the MUSDB18 dataset, as it is smaller in size than the MedleyDB dataset. Subsequently, we performed the same set of experiments on the MedleyDB dataset.

## 4.1 Our Proposed Single Artificial Neural Network Architecture

This section first describes our experimental environment setup, followed by the proposed single ANN architecture, and finally introduces the evaluation metrics used.

We constructed our proposed ANN architecture by conducting preliminary experiments and measuring their results using more precise metrics. This led us to employ PCA or Incremental PCA to build a simple and efficient architecture.

### 4.1.1 Experimental Environment Setup

Our experiments were conducted on a desktop PC equipped with an AMD Ryzen 9 3900X processor and 64 GB of system memory. The Nvidia GeForce RTX 3060 Ti OC graphics card with 8 GB of memory was used for the Artificial Neural Networks (ANNs) experiments, and the Nvidia GeForce RTX 5060 Ti graphics card with 16 GB of memory was used for the Convolutional Neural Networks (CNNs) experiments. TensorFlow[12] was used as the deep learning framework.

### 4.1.2 Model Description

The audio clips in our dataset are longer (1-minute) than those in the previous studies. To utilize the full length of the audio clips, we built a model using the CPU. But our system's 64 GB of memory was insufficient to build a deeper model for better performance. We then built a model using the GPU. Due to the limited memory size on the graphics card, we could not use the full 1-minute audio excerpts. We had to shorten them to 1,400 samples (approximately 3.17 seconds). This showed a better performance. However, we still aimed to utilize the full 1-minute length of the audio clips.

#### Applying PCA on the MUSDB18 Dataset

We then applied Principal Component Analysis (PCA) immediately before feeding the data into the input layer. For the number of PCA components, we first used the default setting, which chooses the minimum number between the number of samples and the number of features. As a result, the number of input nodes was reduced to 586 from 383,600, corresponding to the number of audio clips in the training set. This outperformed the previous models with CPU and GPU, which were trained without PCA. This model was also able to utilize the full 1-minute audio excerpts.

After conducting some experiments to determine the optimal value for the number of PCA components, we found that 90% of the variance showed the best performance. So

[12]TensorFlow, Deep Learning Framework `https://www.tensorflow.org/`.

the number of nodes in the input layer was reduced to 174, corresponding to 90% of the variance on the MUSDB18 dataset. Figure 4.1 illustrates the architecture of our proposed ANN using PCA, and Table 4.1 summarizes the ANN architecture.

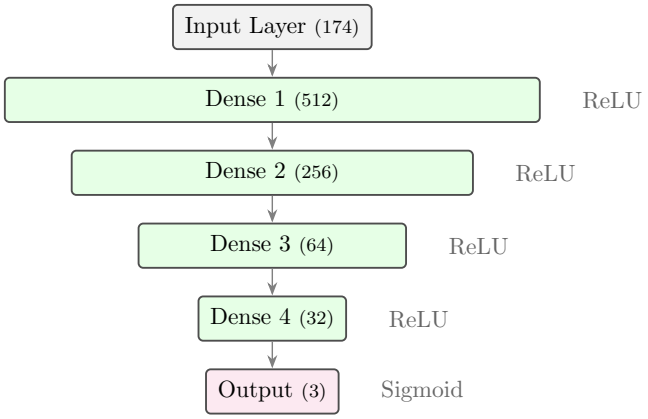



Figure 4.1: Our proposed ANN architecture using PCA.

Table 4.1: The model summary using PCA on the MUSDB18 Dataset.

| **Layer (type)** | **Output Shape** | **Param #** |
|---|---|---|
| Input Layer | 174 | - |
| Dense 1 | 512 | 89,600 |
| Dense 2 | 256 | 131,328 |
| Dense 3 | 64 | 16,448 |
| Dense 4 | 32 | 2,080 |
| Output Layer | 3 | 99 |

We used the ReLU activation function in each layer and the Sigmoid activation function in the output layer. The ADAM optimizer and binary cross-entropy loss function were employed to stabilize learning and minimize prediction errors during training. The learning rate was set to the default value in TensorFlow, which is 0.001.

We also conducted experiments using both Sigmoid activation function with three output nodes and Softmax activation functions with seven output nodes in the output layer. When the Softmax activation function was used, we applied the sparse categorical cross-entropy as the loss function.

### Applying Incremental PCA on the MedleyDB Dataset

Our derived dataset from MedleyDB is much larger than the MUSDB18 dataset. As a result, the PCA implementation in the Scikit-learn library was unable to perform dimensionality reduction because all of the dataset could not be loaded into the system memory, so we instead applied the Incremental PCA from the same library, as described in Section 2.3.2.

To determine an optimal number of components, we first conducted small experiments on a subset of the dataset using standard PCA, since the Incremental PCA does not accept a percentage of variance as an input parameter. From these experiments, we selected 180 as the parameter for the number of variance, which corresponded to 80% of the variance in our derived dataset from MedleyDB.

We built a simple model based on Figure 4.1, and as the size of the data set increased, the model was configured with more layers. Table 4.2 presents a summary of the model employing Incremental PCA.

Table 4.2: The model summary using Incremental PCA on the MedleyDB dataset.

| **Layer (type)** | **Output Shape** | **Param #** |
|---|---|---|
| Input Layer (InputLayer) | 180 | - |
| Dense 1 (Dense) | 4,096 | 741,376 |
| Dense 2 (Dense) | 4,096 | 16,781,312 |
| Dense 3 (Dense) | 2,048 | 8,390,656 |
| Dense 4 (Dense) | 2,048 | 4,196,352 |
| Dense 5 (Dense) | 1,024 | 2,098,176 |
| Dense 6 (Dense) | 1,024 | 1,049,600 |
| Dense 7 (Dense) | 512 | 524,800 |
| Dense 8 (Dense) | 256 | 131,328 |
| Dense 9 (Dense) | 128 | 32,896 |
| Dense 10 (Dense) | 64 | 8,256 |
| Dense 11 (Dense) | 32 | 2,080 |
| Output Layer | 7 | 231 |

We used the Sigmoid activation function and employed the Hamming loss instead of the binary cross-entropy loss because the binary vector for the MedleyDB dataset is larger than the one for the MUSDB18 dataset. All other settings were kept the same as in the previous model using the MUSDB18 dataset.

## 4.2 Empirical Experiments on the MUSDB18 Dataset

This section first describes the experiment preparation. It then presents experiments using a simple model and a hierarchical model.

We first needed to reorganize the dataset because the *other* class was too noisy, meaning it contained too much information. We then conducted experiments using both Sigmoid and Softmax activation functions.

### 4.2.1 Experiment Preparation

We derived a dataset using audio excerpts from each audio track obtained using the MUSDB18 Python package, which are 1-minute segments ending 6 seconds before the end of each track. The excerpt processing is described in Section 3.2.2. This dataset is acoustically simple because it does not include combinations such as bass with vocals or vocals with drums. In addition, each piece of music has relatively well-balanced levels across instruments.

So we generated a synthesized dataset by randomly selecting combinations of instruments from the excerpts of the original dataset, as described in Section 3.2.2. For example, it includes combinations of the vocals from one track with the drums from another, so it might not sound balanced across instruments.

We first used a 4-bit binary vector as the label, in the form of $[vocals, drums, bass, other]$. After conducting some experiments, we found that the *other* class was too noisy because it would contain various instrument families. Table 4.3 shows the model performance with the *other* class, which used the evaluation method in TensorFlow. The *other* class corresponds to any combination of instruments, excluding vocals, drums, and bass. As a result, the loss rate was very high in classes containing the *other* class, making this model very difficult to predict. So we decided to exclude this class. As a result, it has a 3-bit binary vector as the label, in the form of $[vocals, drums, bass]$. The corresponding one-hot encodings are described in Table 4.4.

Table 4.3: Category performance before excluding the *other* class on the MUSDB18 Dataset.

| Class | Accuracy (%) | Loss |
|---|---|---|
| vocals | 96.55 | 4.17 |
| drums | 86.21 | 14.23 |
| bass | 96.55 | 2.03 |
| **other** | **41.38** | **50.13** |
| vocals_drums | 65.52 | 8.18 |
| vocals_bass | 72.41 | 14.66 |
| **vocals_other** | **86.21** | **15.20** |
| drums_bass | 93.10 | 2.22 |
| **drums_other** | **51.72** | **41.05** |
| **bass_other** | **79.31** | **31.89** |
| vocals_drums_bass | 44.83 | 19.92 |
| **vocals_drums_other** | **68.97** | **25.84** |
| **vocals_bass_other** | **72.41** | **28.60** |
| **drums_bass_other** | **41.38** | **37.99** |
| **vocals_drums_bass_other** | **55.17** | **37.51** |
| **Overall** | **70.11** | **22.24** |

Table 4.4: One-hot encodings for simple and hierarchical models on the MUSDB18 dataset.

| Index | Class | Encoding |
|---|---|---|
| 0 | vocals | [1,0,0] |
| 1 | drums | [0,1,0] |
| 2 | bass | [0,0,1] |
| 3 | vocals_drums | [1,1,0] |
| 4 | vocals_bass | [1,0,1] |
| 5 | drums_bass | [0,1,1] |
| 6 | vocals_drums_bass | [1,1,1] |

The dataset was split into training (70%), validation (10%), and test (20%) sets, and this modification was used for the other experiments on the MUSDB18 dataset.

### 4.2.2 Simple Model on the MUSDB18 Dataset

We used the model architecture described in Figure 4.1 and tuned the model by increasing or decreasing a few layers. Table 4.5 shows the best result obtained in this experiment.

The best result was obtained from the model using the Sigmoid activation function, with an accuracy of 72.41%, whereas the experiment including the *other* class in Table 4.3 achieved around 70% using the evaluation method from TensorFlow, which dropped to around 60% when we applied the evaluation method from the Sci-kit Learn library.

The confusion matrix in Figure 4.2 represents the relationship between the predicted

Table 4.5: Precision, recall, and F1-score of the simple model on the MUSDB18 dataset.

| Index | Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|---|
| 0 | vocals | 0.7778 | **0.9655** | 0.8615 | 29 |
| 1 | drums | 0.7037 | 0.6552 | 0.6786 | 29 |
| 2 | bass | **0.8485** | **0.9655** | **0.9032** | 29 |
| 3 | vocals_drums | 0.7692 | 0.6897 | 0.7273 | 29 |
| 4 | vocals_bass | 0.8148 | 0.7586 | 0.7857 | 29 |
| 5 | drums_bass | 0.5405 | 0.6897 | 0.6061 | 29 |
| 6 | vocals_drums_bass | 0.5882 | 0.3448 | 0.4348 | 29 |
| **Accuracy** | | | **0.7241** | | 203 |

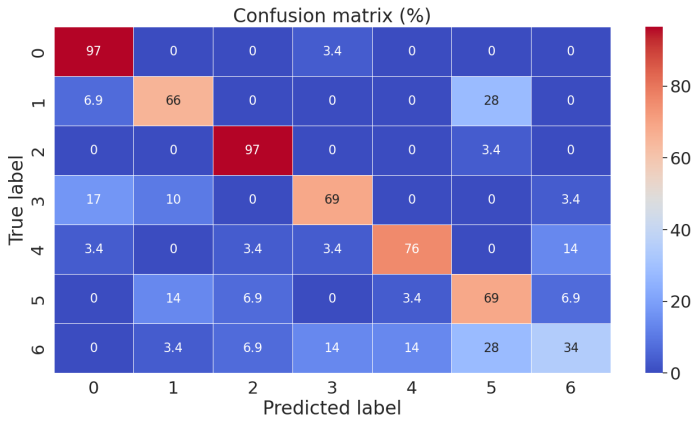



Figure 4.2: Confusion matrix for the simple model.

labels and the truth labels. The performance of this model was better for the *vocals* and *bass* classes, but for the *drums* class, 28% of the audio clips were misclassified as *drums_bass*. This may be due to the imbalance of amplitude between these two classes, as the kick drum often masks the bass guitar in real-world music. The *vocals_drums_bass* class is also spread across the two-instrument classes, indicating that one of the three instruments is often masked by the other two instruments.

### 4.2.3 Hierarchical Model on the MUSDB18 Dataset

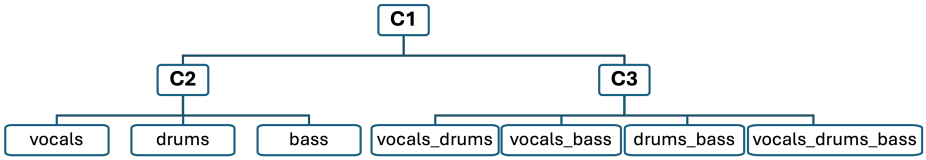
The previous experiment shows that the simple ANN model we proposed is not effective in handling the mixed-instrument classes. So we built hierarchical models, which consist of multiple simple models.

### Approach to Building a Hierarchical Model

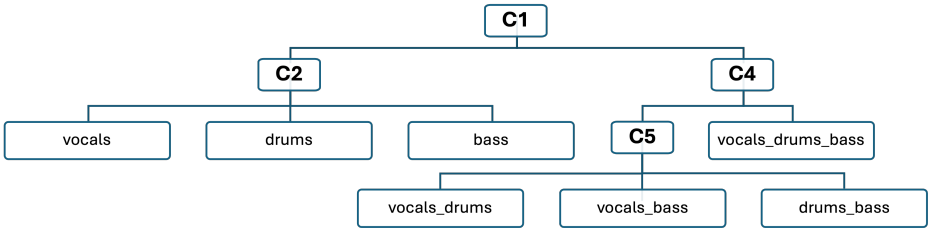
The benefit of applying a hierarchical structure is that each model in the hierarchy can be trained independently, so each model has a smaller, more specific role, unlike a single model that classifies all classes on its own. However, a drawback of the hierarchical structure is that it may become inefficient as the depth of the hierarchy increases because the performance of models at higher levels directly affects the models at lower levels.

We used a simple approach to build hierarchical models. First, we built a model that groups the classes into a smaller number of high-level groups at the top level, aiming for an accuracy higher than 90%. We then applied the same strategy to build models for the subsequent sub-level classifications. In order to train each model, the labels had to be converted to align with the classification goal of each model.

Figure 4.3 illustrates the structure of the hierarchical model 1 and model 2. Table 4.6 shows the results of hierarchical model 1 and model 2, and Table 4.7 shows the accuracies of internal models in each hierarchical structure. Figure 4.4 presents the confusion matrices for the corresponding models.




(a) Model 1

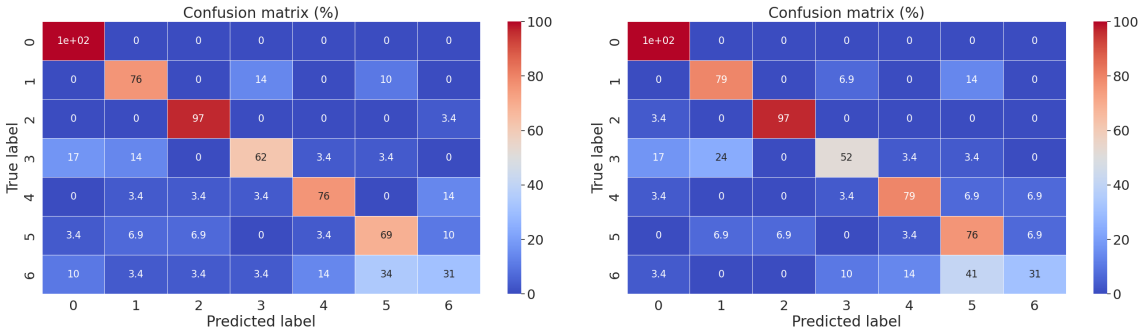



(b) Model 2

Figure 4.3: Structure of the hierarchical model 1 and 2 on the MUSDB18 Dataset.

Table 4.6: Performance comparison between Hierarchical Model 1 and Model 2 on the MUSDB18 dataset.

| Index | Class | Model 1 | | | Model 2 | | |
|---|---|---|---|---|---|---|---|
| | | Precision | Recall | F1-score | Precision | Recall | F1-score |
| 0 | vocals | 0.7632 | **1.0000** | 0.8657 | **0.7838** | **1.0000** | **0.8788** |
| 1 | drums | **0.7333** | 0.7586 | 0.7458 | 0.7188 | **0.7931** | **0.7541** |
| 2 | bass | 0.8750 | **0.9655** | 0.9180 | **0.9333** | **0.9655** | **0.9492** |
| 3 | vocals_drums | **0.7500** | **0.6207** | **0.6792** | 0.7143 | 0.5172 | 0.6000 |
| 4 | vocals_bass | 0.7857 | 0.7586 | 0.7719 | **0.7931** | **0.7931** | **0.7931** |
| 5 | drums_bass | **0.5882** | 0.6897 | **0.6349** | 0.5366 | **0.7586** | 0.6286 |
| 6 | vocals_drums_bass | 0.5294 | **0.3103** | 0.3913 | **0.6923** | **0.3103** | **0.4286** |
| **Accuracy** | | 0.7291 | | | **0.7340** | | |




(a) Confusion Matrix of Model 1

(b) Confusion Matrix of Model 2

Figure 4.4: Confusion matrices for the two hierarchical models.

### Hierarchical Model 1

We first built the hierarchical model 1 in Figure 4.3(a). A model C1 at the top level classifies between pure-instrument and mixed-instrument classes. C1 uses the Sigmoid activation function, and the remaining models in the structure use the Softmax activation function. Figure 4.4(a) shows the confusion matrix of the hierarchical model 1.

The overall accuracy of the model, 72.91%, is slightly higher than that of the simple model, which achieved 72.41%. The accuracies of C1 and C2 are approximately 88% and 99%, respectively, whereas the accuracy of C3 is relatively low at around 70%. We hypothesized that the *vocals_drums_bass* class requires additional treatment to improve performance, as it is the most artificial of all classes, and constructed the next model accordingly.

Table 4.7: Performance of models at each level in the hierarchical structures on the MUSDB18 dataset.

| Model | C1 Acc. (%) | C2 Acc. (%) | C3 Acc. (%) | C4 Acc. (%) | C5 Acc. (%) |
|---|---|---|---|---|---|
| Model 1 | 88 | 99 | 70 | - | - |
| Model 2 | 88 | 99 | - | 79 | 89 |

**Hierarchical Model 2**

We built the hierarchical model 2 to classify the *vocals_drums_bass* class separately. In the model in Figure 4.3(b), C3 from hierarchical model 1 was removed, and C4 and C5 were added, and all models in the hierarchy used the Softmax activation function. Figure 4.4(b) shows the confusion matrix of the hierarchical model 2.

The accuracies of C4 and C5 are approximately 79% and 89%, respectively. Thus, the overall accuracy of the hierarchical model 2 increased slightly to 73.40%, which is the best among our experiments. But the improvement over the previous models is still small. The *vocals_drums_bass* class was still frequently confused with the *drums_bass* class.

C1 was modified to use the Softmax activation function instead of Sigmoid. But it achieved the same accuracy. However, the performance for each class was changed. The *vocals_drums* class was more often misclassified as the *drums* class.

From the experiments comparing the simple model and the hierarchical models, we experienced both benefits and drawbacks of the hierarchical structure. Although the results were not as good as expected, the experiments conducted on the MUSDB18 dataset provided a solid foundation for the subsequent experiments of our work with the MedleyDB dataset.

## 4.3 Empirical Experiments on the MedleyDB Dataset

In this section, we first describe the experiment preparation, and then we present experiments using a simple model and a hierarchical model. Before conducting experiments on the MedleyDB dataset, we needed to reorganize the dataset because it contains a large number of instrument classes but contains only a small number of audio stems for each.

### 4.3.1 Experiment Preparation

The MedleyDB dataset contains 84 instruments in the audio stems. But many of them have very few samples. Therefore, we first grouped these instruments into ten (10) classes (*bass*, *drums*, *guitars*, *piano*, *strings*, *vocals*, *woodwinds*, *synths*, *brass*, and *keyboards*), and conducted a simple experiment to classify these ten (10) classes using a simple ANN model. The results were quite poor, with an accuracy of only about 44%, so we subsequently excluded three instrument classes from the dataset and performed upsampling.

#### Dataset Reorganization

The final dataset consisted of seven classes after excluding three classes from the ten classes. Table 4.8 presents the overview of the reorganized dataset.

Table 4.8: Instrument categories and counts in MedleyDB dataset.

| Class | Instrument | # of audio clips | Total |
|---|---|---|---|
| bass | electric bass | 84 | 84 |
| drums | auxiliary percussion | 25 | 152 |
| | drum set | 107 | |
| | drum machine | 20 | |
| guitars | clean electric guitar | 80 | 120 |
| | acoustic guitar | 40 | |
| piano | piano | 72 | 72 |
| strings | violin section | 26 | 150 |
| | violin | 48 | |
| | viola | 16 | |
| | cello | 21 | |
| | double bass | 39 | |
| vocals | male singer | 61 | 151 |
| | female singer | 52 | |
| | vocalists | 38 | |
| woodwinds | flute | 22 | 54 |
| | bassoon | 14 | |
| | clarinet | 18 | |
| | Grand Total | 783 | 783 |

To reduce the number of classes, we first excluded all instruments with 10 or fewer audio samples. As a result, we excluded the *keyboards* class. We also excluded the *brass* class because it contained only a total of 16 audio samples, which is much smaller than the number of audio samples in other classes.

We then excluded the *synth* instrument, as it is similarly ambiguous to the *other* class in the MUSDB18 dataset. Synthesizers generate sound using oscillators or wavetables, which can produce sounds similar to drums, bass, lead, piano, and pad instruments [61]. So the labels for the *synth* instrument should also be more specific, such as synth bass, synth piano, and synth drums. However, in the MedleyDB dataset, these sounds are labelled only as synthesizer. So we excluded the *synth* instrument. We also excluded the distorted guitar instrument for the same reason as the synthesizer; their sound spectra are too wide.

The double bass is categorized as *strings* class. It could be regarded as the *bass* class due to its very low pitch, and it is typically played by plucking the strings with the fingers in jazz music, producing a similar sound to a fretless bass guitar. However, we regard it as part of the string family instruments because of its timbre, as the MedleyDB dataset also includes some classical music.

Since there are seven pure-instrument classes, the total number of possible combinations is $2^7 - 1 = 127$, which is excessively large. Therefore, we selected only the most desirable and reasonable combinations. Table 4.9 presents the resulting combinations used in this study. For the rest of our experiments on the MedleyDB dataset, we used a 7-bit binary vector as the label, with the corresponding one-hot encodings listed in Table 4.10.

Table 4.9: Class combinations derived from the MedleyDB dataset.

| **# of Instruments** | **Class Combinations** |
|---|---|
| 1 instrument | bass, drums, guitars, piano, strings, vocals, woodwinds |
| 2 instruments | bass_drum |
| 3 instruments | bass_drum_guitars, bass_drum_piano, bass_drum_strings<br>bass_drum_vocals, bass_drum_woodwinds |
| 4 instruments | bass_drum_guitars_piano, bass_drum_guitars_vocals<br>bass_drum_piano_strings, bass_drum_piano_vocals<br>bass_drum_strings_vocals |

We included the *others* class, which comprises binary vectors that are not assigned to any of the other classes. This issue did not arise in the experiments on the MUSDB18 dataset because it contains seven classes, and a 3-bit representation was sufficient, since

Table 4.10: One-hot encodings for experiments on the MedleyDB dataset.

| Index | Class | Encoding |
|---|---|---|
| 0 | bass | [1, 0, 0, 0, 0, 0, 0] |
| 1 | drums | [0, 1, 0, 0, 0, 0, 0] |
| 2 | guitars | [0, 0, 1, 0, 0, 0, 0] |
| 3 | piano | [0, 0, 0, 1, 0, 0, 0] |
| 4 | strings | [0, 0, 0, 0, 1, 0, 0] |
| 5 | vocals | [0, 0, 0, 0, 0, 1, 0] |
| 6 | woodwinds | [0, 0, 0, 0, 0, 0, 1] |
| 7 | bass_drums | [1, 1, 0, 0, 0, 0, 0] |
| 8 | bass_drums_guitars | [1, 1, 1, 0, 0, 0, 0] |
| 9 | bass_drums_piano | [1, 1, 0, 1, 0, 0, 0] |
| 10 | bass_drums_strings | [1, 1, 0, 0, 1, 0, 0] |
| 11 | bass_drums_vocals | [1, 1, 0, 0, 0, 1, 0] |
| 12 | bass_drums_woodwinds | [1, 1, 0, 0, 0, 0, 1] |
| 13 | bass_drums_guitars_piano | [1, 1, 1, 1, 0, 0, 0] |
| 14 | bass_drums_guitars_vocals | [1, 1, 1, 0, 0, 1, 0] |
| 15 | bass_drums_piano_strings | [1, 1, 0, 1, 1, 0, 0] |
| 16 | bass_drums_piano_vocals | [1, 1, 0, 1, 0, 1, 0] |
| 17 | bass_drums_strings_vocals | [1, 1, 0, 0, 1, 1, 0] |
| 18 | others | All other combinations |

$2^3 - 1 = 7$.

**Class Balancing via Upsampling and Audio Data Augmentation**

The number of audio clips in each class was imbalanced. In particular, the *woodwinds* class contained only 54 tracks. Therefore, we applied upsampling techniques to increase the number of audio clips to balance across the classes.

We used one excerpt from each audio track in the MUSDB18 dataset using the way described in Section 3.2.2. In contrast, for the experiments on the MedleyDB dataset, we used both the first and second halves of each audio clip, which doubled the number of audio clips.

We then doubled the number of audio clips by using alternative parameter settings for the Constant-Q Transform (CQT) and Chroma Energy Normalized Statistics (CENS) audio features. By default, CQT uses L1 normalization and CENS uses L2 normalization. To increase the number of samples, we set the normalization parameter to $\infty$ for both feature extraction functions. So that the maximum value becomes 1 and all other values are scaled

proportionally.

Lastly, we added Gaussian noise to these audio clips, as described in Section 3.2.2, which further doubled the size of the dataset. We aimed to obtain 500 audio clips for each class. However, the *woodwinds* class still did not reach this size, so we generated more audio clips with Gaussian noise.

After these steps, the dataset was divided into training (80%) and test (20%) sets, containing 400 and 100 audio clips, respectively. We conducted small experiments with different train–validation–test splits, including K-fold cross-validation, but the validation set did not provide additional benefits for experiments with our derived dataset. So we adopted only a train–test split.

### 4.3.2 Simple Model on the MedleyDB Dataset

This experiment consists of two parts: the first part focused on classifying only pure-instrument classes, and the second part focused on classifying all classes. We used the model architecture with Incremental PCA, described in Section 4.1.2.

#### Classifying Pure-instrument Classes using a Simple Model

We first conducted experiments to classify the pure-instrument classes. Table 4.11 and Figure 4.5 present the results and the corresponding confusion matrix, respectively.

Table 4.11: Precision, recall, and F1-score of the simple model on the MedleyDB dataset (pure-instrument subset).

| Index | Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|---|
| 0 | bass | **0.9074** | **0.9800** | **0.9423** | 100 |
| 1 | drums | 0.7500 | 0.6000 | 0.6667 | 100 |
| 2 | guitars | 0.7714 | 0.5400 | 0.6353 | 100 |
| 3 | piano | 0.6452 | 0.8000 | 0.7143 | 100 |
| 4 | strings | 0.5691 | 0.7000 | 0.6278 | 100 |
| 5 | vocals | 0.8919 | 0.6600 | 0.7586 | 100 |
| 6 | woodwinds | 0.7547 | 0.8000 | 0.7767 | 100 |
| **Accuracy** | | | **0.7257** | | 700 |

The overall accuracy was 72.57%, and 15 audio clips were misclassified, corresponding to 2.14% of the test set. In all of these cases, the model incorrectly predicted that the clip

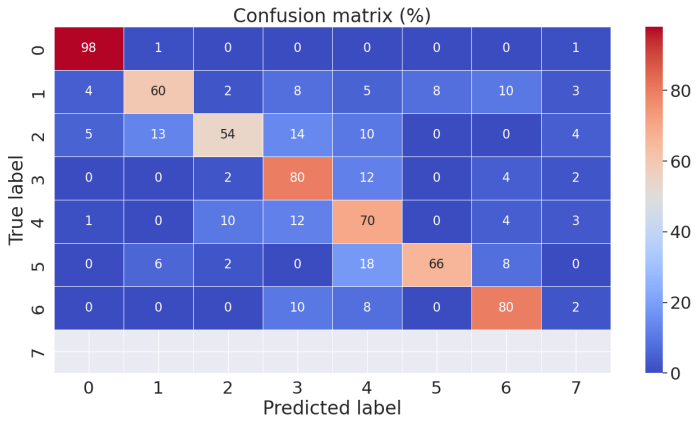



Figure 4.5: Confusion matrix for the simple model on the MedleyDB dataset (pure-instrument subset).

contained more than one instrument, although only one instrument was present. The result may not be poor, given that the seven (7) classes actually represent 18 distinct kinds of instrument stems.

The *drum* and *guitars* classes were often misclassified as other classes, whereas many classes tended to be predicted as the *piano* and *string* classes. Auxiliary percussion within the *drum* class may have caused misclassification as other classes. The *vocals* class was often misclassified as *strings* or *woodwinds* classes, which is understandable because they overlap frequency ranges and have a similarity in timbre when vocals produce sustained vowel sounds.

### Classifying All Classes using a Simple Model

We extended the experiment by including the remaining classes so that the model can classify both pure-instrument and mixed-instrument classes. Table 4.12 and Figure 4.6 present the classification results and the corresponding confusion matrix, respectively.

The result was worse than expected, with an overall accuracy of 34.56%. The classification performance for pure-instrument classes remained reasonably acceptable. However, almost all metrics decreased substantially.

Starting from the *bass_drums* class, the model became confused and frequently misclas-

Table 4.12: Precision, recall, and F1-score of the simple model on the MedleyDB dataset (all instruments).

| Index | Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|---|
| 0 | bass | 0.5672 | 0.7600 | 0.6496 | 100 |
| 1 | drums | 0.5270 | 0.3900 | 0.4483 | 100 |
| 2 | guitars | 0.5698 | 0.4900 | 0.5269 | 100 |
| 3 | piano | 0.4079 | 0.6200 | 0.4921 | 100 |
| 4 | strings | 0.7143 | 0.6000 | 0.6522 | 100 |
| 5 | vocals | 0.6931 | 0.7000 | 0.6965 | 100 |
| 6 | woodwinds | 0.7931 | 0.6900 | 0.7380 | 100 |
| 7 | bass_drums | 0.2081 | 0.4100 | 0.2761 | 100 |
| 8 | bass_drums_guitars | 0.1522 | 0.1400 | 0.1458 | 100 |
| 9 | bass_drums_piano | 0.1389 | 0.3000 | 0.1899 | 100 |
| 10 | bass_drums_strings | 0.2273 | 0.1500 | 0.1807 | 100 |
| 11 | bass_drums_vocals | 0.1835 | 0.2000 | 0.1914 | 100 |
| 12 | bass_drums_woodwinds | 0.2955 | 0.1300 | 0.1806 | 100 |
| 13 | bass_drums_guitars_piano | 0.2041 | 0.2000 | 0.2020 | 100 |
| 14 | bass_drums_guitars_vocals | 0.2195 | 0.1800 | 0.1978 | 100 |
| 15 | bass_drums_piano_strings | 0.1930 | 0.1100 | 0.1401 | 100 |
| 16 | bass_drums_piano_vocals | 0.1702 | 0.0800 | 0.1088 | 100 |
| 17 | bass_drums_strings_vocals | 0.1842 | 0.0700 | 0.1014 | 100 |
| **Accuracy** | | | **0.3456** | | 1800 |

sified audio clips into many other classes. This may be due to the fact that these clips are mixed using randomly selected pure-instrument stems, so the balance among instruments is not preserved as in typical music, meaning that the audio clips in our derived dataset sound more artificial than real-world music. As a result, some instruments may strongly mask others, so the model may classify only one or two dominant instruments.

### 4.3.3 Hierarchical Model on the MedleyDB Dataset

We used the same approach as in Section 4.2.3 to build hierarchical models on the MedleyDB dataset. The hierarchical models consist of multiple simple models used in the previous section. We successfully constructed a model for the pure-instrument classes. However, we were unable to obtain a satisfactory model for the mixed-instrument classes due to their low performance.

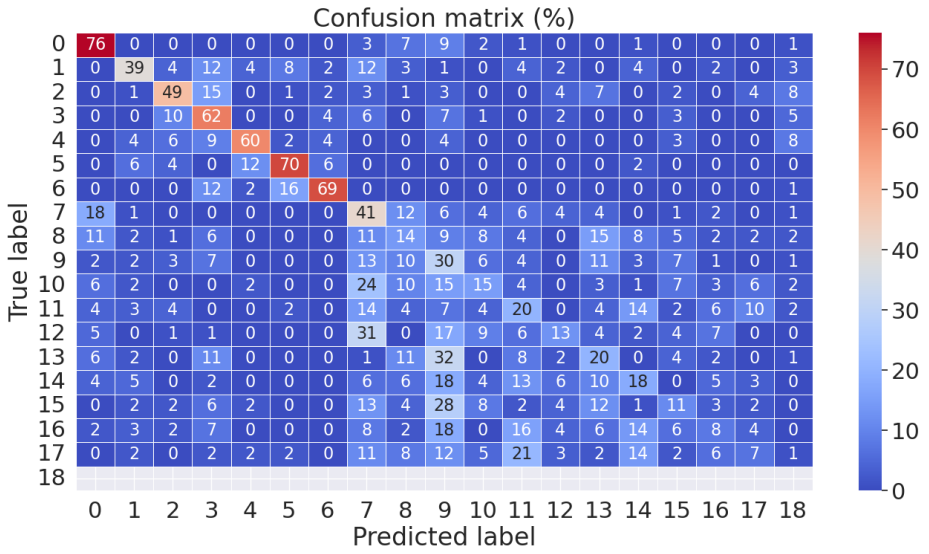


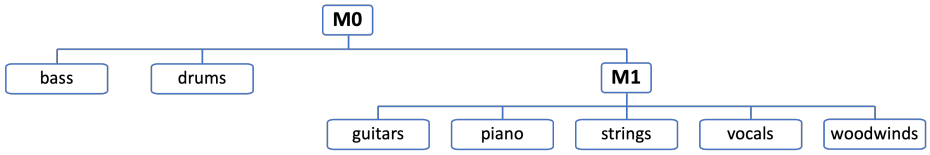

Figure 4.6: Confusion matrix for the simple model on the MedleyDB dataset (all instruments).

### Classifying Pure-instrument Classes using a Hierarchical Model

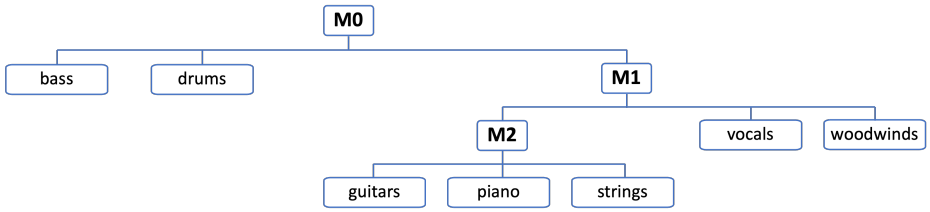
We first built hierarchical models for classifying pure-instrument classes. We selected the model with over 90% accuracy at the top level through extensive trial and error. After obtaining a model with satisfactory accuracy for the top level, we then trained models for the subsequent levels. In total, nine models were developed for pure-instrument classification, from which the top four are presented in this section. Figure 4.7 illustrates the structures of the top four models.

Model 2 has a two-level hierarchical structure, whereas the remaining models have a three-level hierarchical structure. Model 2, Model 3, and Model 4 share the same internal model for classifying the *bass* and *drum* classes, and Model 2 and Model 7 share the same internal model for classifying the *guitars*, *piano*, *strings*, *vocals*, and *woodwinds* classes. Table 4.13 reports the accuracies of internal models in each hierarchical structure, and Table 4.14 presents the classification results of all four hierarchical structures.

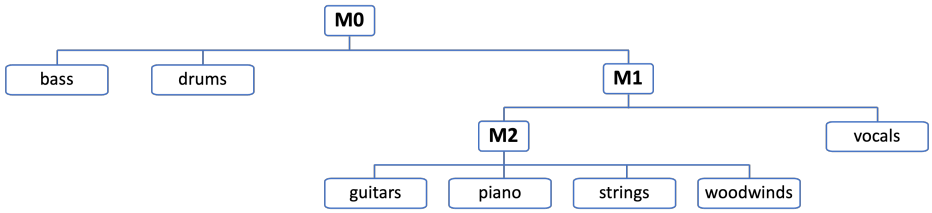
All of them showed better accuracies than the simple model introduced in Section 4.3.2 to classify pure-instrument classes, which achieved an accuracy of 72.57%. The performance of Model 2 was the best among them, with an accuracy of 75.71% and an F1-score



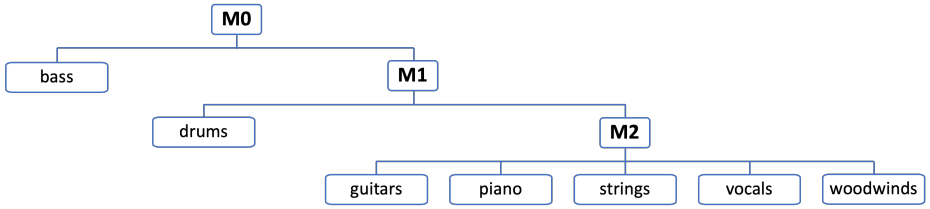

(a) Model 2




(b) Model 3




(c) Model 4




(d) Model 7

Figure 4.7: Structure of the hierarchical models on the MedleyDB Dataset.

of 77. This is because Model 2 has fewer hierarchy than the other models.

The remaining models have reasonably accurate internal models at the second level, and the first-level internal model of Model 7 achieves the highest accuracy. However, since their accuracies are below 100%, classification errors accumulate and propagate to the last level of their hierarchies.

Model 3 achieved a slightly lower accuracy (75.14%) than Model 2 (75.71%), but it showed the smallest error; it rarely classified audio clips to labels that were not seen during training. Figure 4.8 shows the confusion matrices for the four hierarchical models.

As the simple model did, all hierarchical models classified the *bass* class well, whereas

Table 4.13: Performance of models at each level in the hierarchical structures on the MedleyDB dataset.

| **Model** | **M0 Acc. (%)** | **M1 Acc. (%)** | **M2 Acc. (%)** |
|---|---|---|---|
| Model 2 | 92.00 | 75.60 | - |
| Model 3 | 92.00 | 89.60 | 74.33 |
| Model 4 | 92.00 | 93.80 | 76.00 |
| Model 7 | 97.57 | 92.67 | 75.60 |

Table 4.14: Comprehensive Comparison of Precision (P), Recall (R), and F1-score (F1) for Models 2, 3, 4, and 7.

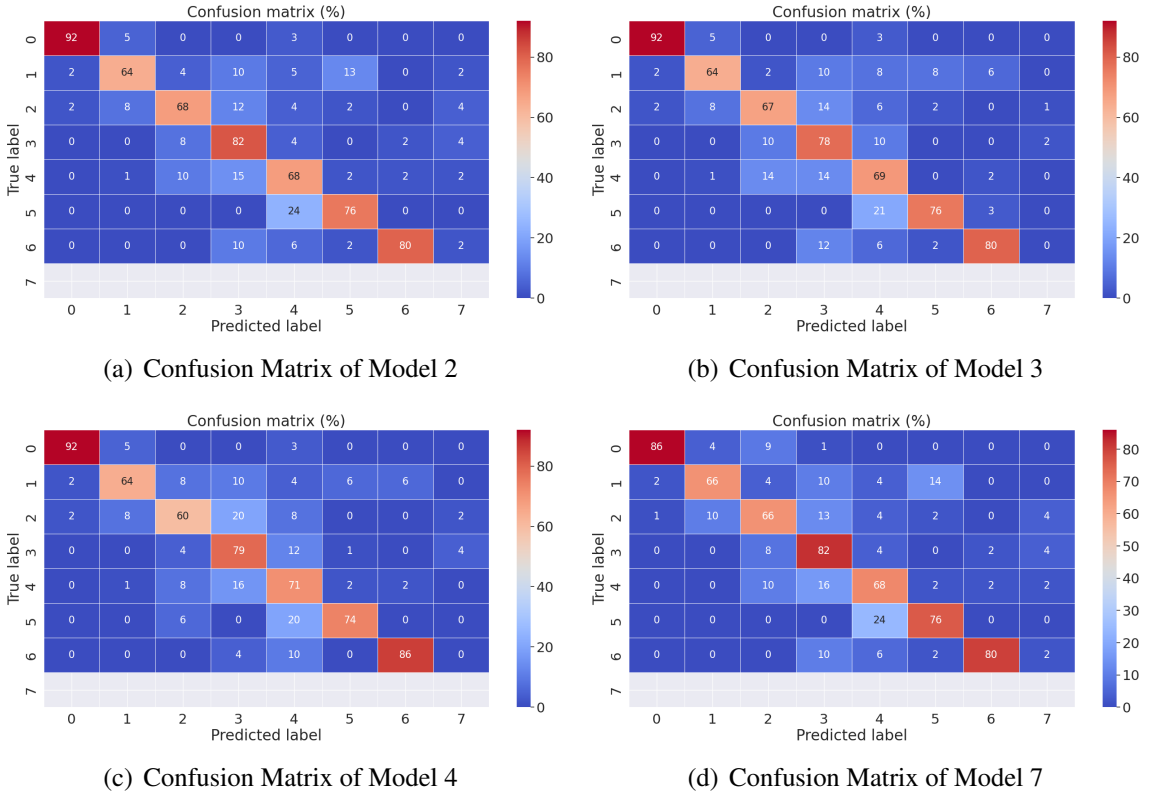
| **Index** | **Class** | **M2** | | | **M3** | | | **M4** | | | **M7** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **P** | **R** | **F1** | **P** | **R** | **F1** | **P** | **R** | **F1** | **P** | **R** | **F1** |
| 0 | Bass | 0.9583 | **0.9200** | **0.9388** | 0.9583 | **0.9200** | **0.9388** | 0.9583 | **0.9200** | **0.9388** | **0.9663** | 0.8600 | 0.9101 |
| 1 | Drums | 0.8205 | 0.6400 | 0.7191 | 0.8205 | 0.6400 | 0.7191 | 0.8205 | 0.6400 | 0.7191 | **0.8250** | **0.6600** | **0.7333** |
| 2 | Guitars | **0.7556** | **0.6800** | **0.7158** | 0.7204 | 0.6700 | 0.6943 | 0.6977 | 0.6000 | 0.6452 | 0.6804 | 0.6600 | 0.6701 |
| 3 | Piano | **0.6357** | **0.8200** | **0.7162** | 0.6094 | 0.7800 | 0.6842 | 0.6124 | 0.7900 | 0.6900 | 0.6212 | **0.8200** | 0.7069 |
| 4 | Strings | 0.5965 | 0.6800 | 0.6355 | 0.5610 | 0.6900 | 0.6188 | 0.5547 | **0.7100** | 0.6228 | **0.6182** | 0.6800 | **0.6476** |
| 5 | Vocals | 0.8000 | **0.7600** | 0.7795 | 0.8636 | **0.7600** | **0.8085** | **0.8916** | 0.7400 | **0.8087** | 0.7917 | **0.7600** | 0.7755 |
| 6 | Woodwinds | **0.9524** | 0.8000 | 0.8696 | 0.8791 | 0.8000 | 0.8377 | 0.9149 | **0.8600** | **0.8866** | **0.9524** | 0.8000 | 0.8696 |
| **Accuracy** | | | **0.7571** | | | 0.7514 | | | 0.7514 | | | 0.7486 | |

the *drums* and *guitars* classes were still often misclassified, with only modest improvement. Many classes still tended to be predicted as *piano* or *string*. But the performance for the *vocals* class was better than in the simple model.

Model 4 achieved the highest precision for the *vocals* class, and because of the separate internal model for the *bass* and *drums* classes in Model 7, those classes showed the highest precision.

### Classifying All Classes using a Hierarchical Model

After the previous experiments, we attempted to train a model at the top level for all classes, but this task was not successful. Classifying all classes into three groups (pure-instrument, 2–3-instrument, and 4-instrument classes) yielded an accuracy of approximately 50%, while classifying them into two groups (pure-instrument vs. mixed-instrument classes) achieved about 83% accuracy. Distinguishing between 2–3-instrument and 4-instrument classes resulted in an accuracy of 52%. We also tried various alternative




(a) Confusion Matrix of Model 2

(b) Confusion Matrix of Model 3

(c) Confusion Matrix of Model 4

(d) Confusion Matrix of Model 7

Figure 4.8: Confusion matrices for the four hierarchical models.

groupings of the classes, such as a separated group consisting of the *bass*, *drums*, and *piano* classes and grouping the remaining classes. But these experiments were also unsuccessful.

Based on these results, we decided not to conduct further experiments on hierarchical models for all classes. Because hierarchical models for all classes would require more levels than the one for pure-instrument classes, their performance would likely degrade substantially due to accumulated errors and poor performance at the upper levels. It might be possible to build a model whose performance on all classes exceeds the accuracy of the simple model (34.56%). However, we concluded that further exploration in this direction would not be particularly meaningful.

## 4.4 Summary

In this chapter, we presented experiments conducted on the MUSDB18 and the MedleyDB datasets. Since the MUSDB18 dataset is smaller than the MedleyDB dataset, we first performed experiments on the MUSDB18 dataset. Building on the experience gained from the experiments with the MUSDB18 dataset, we subsequently conducted experiments on the MedleyDB dataset.

We also introduced additional processes not covered in Chapter 3, such as dataset reorganization, PCA, and Incremental PCA. The Hamming loss function was employed only in the experiments on the MedleyDB dataset.

Training a simple neural network took approximately two to three minutes for the MUSDB18 dataset and ten to fifteen minutes for the MedleyDB dataset, depending on the number of layers. This implies that training each model in the hierarchical structures required substantially more time, depending on the number of models involved.

We demonstrated that hierarchical structures outperformed simple models by splitting the classification task. However, a trade-off arose between the depth of the hierarchy and the overall accuracy due to error propagation. In the experiments on the MUSDB18 dataset, the hierarchical model achieved slightly higher accuracy (73.40%) than the simple model (72.41%), which is not a big difference. But this motivated further experiments on the MedleyDB dataset. In the experiments on pure-instrument classes with the MedleyDB dataset, the simple model reached an accuracy of 72.57%, whereas the best-performing hierarchical model among nine models achieved 75.71%. Although these achievements are modest, they are meaningful because they demonstrate that achieving high performance in musical instrument classification still remains a challenging problem.

It did not work well when we attempted to build a hierarchical model for classifying all classes in the MedleyDB dataset. The simple model for classifying all classes achieved an accuracy of 34.56%, and further experiments to train models at a high level in the hierarchical structure were unsuccessful. While classifying between pure-instrument

and mixed-instrument classes achieved an accuracy of 83%, subsequent experiments with various grouping strategies did not yield sufficient results for a hierarchical structure.

We found that a common issue across the experiments was that some instruments are likely masked by others in mixed-instrument classes, which may lead to degraded performance. There should be methods to address this problem. However, our models using ANNs might not have a strong ability to do so. The artificial characteristics of our derived datasets may also have contributed to the lower performance. So we anticipate that our approach would be more effective when applied directly to real-world music.

# Chapter 5

# Classifying Musical Instruments using Convolutional Neural Networks

This chapter will present empirical experiments on deep learning models employing Convolutional Neural Networks (CNNs) using two datasets, MUSDB18 and MedleyDB. In Chapter 4, we examined the capabilities and limitations of Artificial Neural Networks (ANNs) through experimental evaluations on the same datasets. The experiments presented in this chapter provide insight into the advantages of utilizing CNNs over ANNs.

As we did in Chapter 4, we first conducted experiments using each audio feature on the MUSDB18 dataset and then performed the same experiments on the MedleyDB dataset. Additionally, we conducted further experiments using various combinations of audio features on the MedleyDB dataset.

## 5.1 Our Proposed Convolutional Neural Network Architecture

This section first describes the CNN model architecture. We then explain how to convert the audio features into images. We also introduce the dB-MFCCs, a rescaled variant of Mel-frequency Cepstral Coefficients (MFCCs). Finally, we present the experiments and their evaluations in detail.

### 5.1.1 Model Description

As mentioned in Section 2.1.4, CNNs typically consist of convolution and pooling operations. Figure 5.1 illustrates the architecture of our proposed CNN, and Table 5.1 sum-

marizes the CNN architecture with Mel-spectrograms as input.

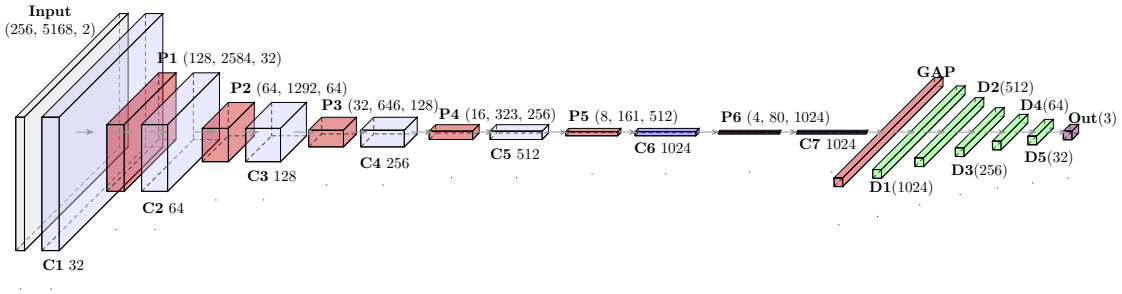



Figure 5.1: Our proposed CNN architecture with Mel-spectrogram input.

Table 5.1: Summary of our proposed CNN architecture with Mel-spectrogram input.

| **Layer (Type)** | **Kernel** | **Output Shape** | **Param #** |
|---|---|---|---|
| Input Layer | - | (256, 5168, 2) | - |
| Conv2D_1 | $3 \times 3$ | (256, 5168, 32) | 608 |
| MaxPooling2D_1 | $2 \times 2$ | (128, 2584, 32) | - |
| Conv2D_2 | $3 \times 3$ | (128, 2584, 64) | 18,496 |
| MaxPooling2D_2 | $2 \times 2$ | (64, 1292, 64) | - |
| Conv2D_3 | $3 \times 3$ | (64, 1292, 128) | 73,856 |
| MaxPooling2D_3 | $2 \times 2$ | (32, 646, 128) | - |
| Conv2D_4 | $3 \times 3$ | (32, 646, 256) | 295,168 |
| MaxPooling2D_4 | $2 \times 2$ | (16, 323, 256) | - |
| Conv2D_5 | $3 \times 3$ | (16, 323, 512) | 1,180,160 |
| MaxPooling2D_5 | $2 \times 2$ | (8, 161, 512) | - |
| Conv2D_6 | $3 \times 3$ | (8, 161, 1024) | 4,719,616 |
| MaxPooling2D_6 | $2 \times 2$ | (4, 80, 1024) | - |
| Conv2D_7 | $3 \times 3$ | (4, 80, 1024) | 9,438,208 |
| GlobalAvgPool | - | 1024 | - |
| Dense_1 | 1024 | 1024 | 1,049,600 |
| Dense_2 | 512 | 512 | 524,800 |
| Dense_3 | 256 | 256 | 131,328 |
| Dense_4 | 64 | 64 | 16,448 |
| Dense_5 | 32 | 32 | 2,080 |
| Output | 3 | 3 | 99 |

Our model uses padding with a $1 \times 1$ stride so that the output of each convolution with a $3 \times 3$ filter has the same size as its input, and max-pooling operation with a $2 \times 2$ kernel and a $2 \times 2$ stride halves the dimensions of the images. Each convolutional and hidden layer uses the ReLU activation function, while the output layer employs the Sigmoid activation function.

The size of the input layer varies depending on the audio feature used, which affects the number of convolutional layers. Since there is a limit to halve the image before the images become the size of $1 \times 1$, this constraint leads to fewer layers of networks. So we controlled the number of the last few convolutional layers in our study depending on the size of the audio feature image. We introduce the audio features employed for the CNNs in the next subsection.

### 5.1.2 Image-Based Representations of Audio Features and dB-MFCCs

We employed four audio features, such as Mel-spectrogram, Mel-Frequency Cepstral Coefficients (MFCCs), Constant-Q Transform (CQT), and Chroma Energy Normalized Statistics (CENS). The zero-crossing rate was not included as it is a one-dimensional temporal feature and cannot be transformed into a two-dimensional image representation compatible with our CNNs. Table 5.2 summarizes the input shapes of the audio features.

Table 5.2: Input shapes of audio features for CNN-based models.

| **Feature Type** | **Input Shape** |
|---|---|
| Mel-spectrogram | $256 \times 5618 \times 2$ |
| MFCCs | $40 \times 5618 \times 2$ |
| CQT | $80 \times 5618 \times 2$ |
| CENS | $12 \times 5618 \times 2$ |

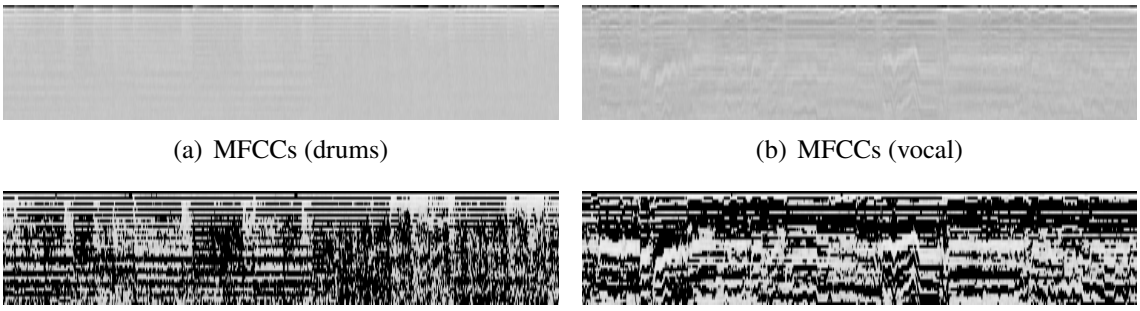
Their detailed audio feature extraction process is described in Section 3.2.3. Each extracted audio feature is converted into two separate single-channel grayscale PNG image files, one for each channel, since we use a stereo channel. Then, when each audio clip is fed into a CNN, the two image files are combined into a single two-channel input data. Thus, each audio feature image has the same length and the same number of channels but differs in height.

#### dB-scaled MFCCs

Audio signal compression has many applications [52] and is also widely used in music production. Although it may appear to simply correct the dynamic range of the amplitude, it

actually changes the characteristics of the audio signal, making weak or masked harmonics of sound more prominent. When applied to grouped signals in a mixer, it also helps to blend sounds in a more pleasing and cohesive way [27].

We assumed that these characteristics of the compression could also be applied to the audio features, particularly MFCCs. As MFCCs are a low-dimensional representation of the Mel-spectrogram, they are not directly interpreted as the loudness or physical amplitude of the audio signal. However, from a data perspective, enhancing low-energy components can reveal additional information in the data that may be detected by CNNs. Therefore, we reshaped the MFCC curves on a logarithmic scale by applying the `power_to_db` method from the librosa Python library so that it makes patterns more accessible to the CNNs. We refer to these transformed features as dB-MFCCs. Figure 5.2 shows a visual comparison between the original MFCCs and the proposed dB-MFCCs.



(a) MFCCs (drums) (b) MFCCs (vocal)

(c) dB-MFCCs (drums) (d) dB-MFCCs (vocal)

Figure 5.2: Visual comparison between MFCCs and dB-MFCCs.

Figure 5.2(a) and (b) show drum and vocal images of the original MFCCs, respectively, while Figure 5.2(c) and (d) show the corresponding images of the proposed dB-MFCCs. In the images of the original MFCCs, the patterns are difficult to recognize, whereas in the images of the dB-MFCCs, the patterns become more clearly visible. We conducted a small experiment using our proposed CNN architecture to compare these two features, and the results are presented in Table 5.3.

Our proposed dB-MFCCs show better performance, achieving an accuracy of 87.68%, compared to 85.22% for the original MFCCs. This improvement is particularly noticeable

Table 5.3: Performance comparison between original MFCCs (Model 1) and our proposed dB-MFCCs (Model 2).

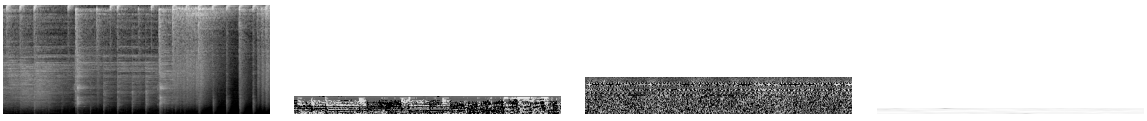
| Index | Class | Model 1 (MFCC) | | | Model 2 (dB-MFCC) | | |
|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 |
| 0 | vocals | 0.8571 | 0.8276 | 0.8421 | **0.9655** | **0.9655** | **0.9655** |
| 1 | drums | 0.8182 | **0.9310** | **0.8710** | **0.8889** | 0.8276 | 0.8571 |
| 2 | bass | 0.9655 | **0.9655** | **0.9655** | **1.0000** | 0.8966 | 0.9455 |
| 3 | vocals_drums | 0.7812 | **0.8621** | 0.8197 | **0.8846** | 0.7931 | **0.8364** |
| 4 | vocals_bass | 0.8571 | 0.8276 | 0.8421 | **0.8667** | **0.8966** | **0.8814** |
| 5 | drums_bass | **0.8800** | 0.7586 | 0.8148 | 0.7778 | **0.9655** | **0.8615** |
| 6 | vocals_drums_bass | **0.8214** | **0.7931** | **0.8070** | 0.7931 | **0.7931** | 0.7931 |
| **Accuracy** | | | 0.8522 | | | **0.8768** | |

for the mixed-instrument classes. Although the difference between their best accuracies was only 2.46%, the model with dB-MFCCs typically achieved about 5–6% higher accuracy. While the model using the original MFCCs sometimes dropped to accuracies in the mid-60% range, the model using the dB-MFCCS never fell below the mid-70% range.

Based on our experiments, we decided to adopt the proposed dB-MFCCs as our default MFCC representation. Hereafter, unless otherwise specified, the term MFCCs refers to dB-MFCCs.

We also applied this method to CQT and CENS. However, we did not focus much on these features since we used only the real part and discarded the imaginary part of the CQT. CENS also provides only 12 dimensions, so we did not expect noticeable performance.

**Visual Examples of Audio Features**

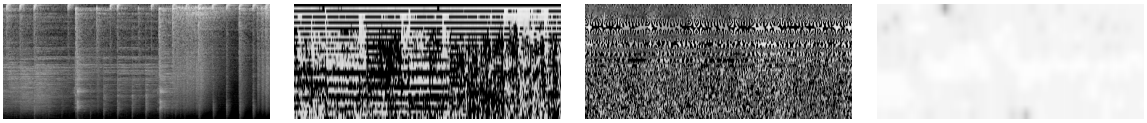


(a) Mel-spectrogram (b) dB-MFCCs (c) CQT (d) CENS

Figure 5.3: Visual comparison of audio features for drums at the original scale.

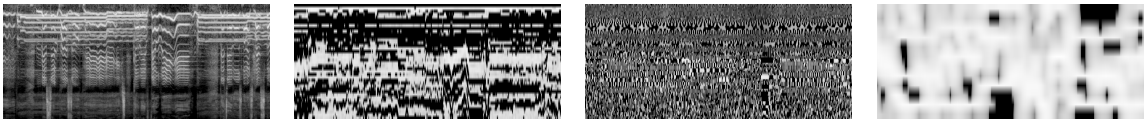
Figure 5.3 shows a part of the images converted from each audio feature, which are clipped segments from the full images. Since the features have different shapes, their image heights differ. In particular, the CENS image has a height of only 12 pixels.

We rescaled them to the same height only for visualization purposes, not for feeding them into the CNNs. Figures 5.4 and 5.5 show the resulting feature images for drums and vocals, respectively.



(a) Mel-spectrogram (b) db-MFCCs (c) CQT (d) CENS

Figure 5.4: Visual comparison of audio features for drums after rescaling.



(a) Mel-spectrogram (b) dB-MFCCs (c) CQT (d) CENS

Figure 5.5: Visual comparison of audio features for vocal after rescaling.

The CQT images do not look like typical musical representations, such as spectrograms. Because we discarded the imaginary part of the CQT and thus lost phase information, the resulting images look similar to a PDF417-style barcode, like the one on the back of a driver's license.

## 5.2 Empirical Experiments on the MUSDB18 Dataset

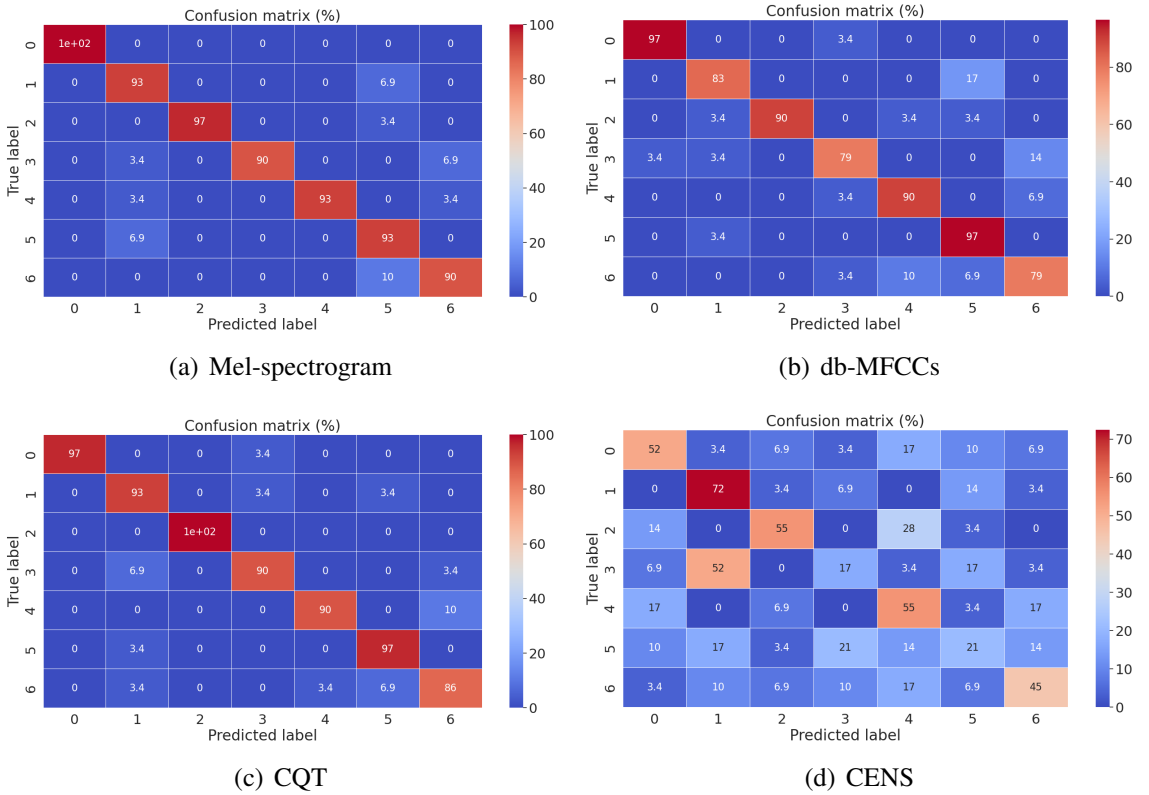
### 5.2.1 Experiments on the MUSDB18 with Audio Feature Images

After obtaining the audio feature images described in the previous section, we conducted experiments using the model and configurations mentioned in Section 5.1. Table 5.4 shows the results of four models, each of which uses a different audio feature image as input, and Figure 5.6 presents the confusion matrices for each corresponding model.

The results exceeded our expectations. Except for the model using the CENS, the models using CNNs outperformed all models using ANNs. The best accuracy was achieved by the model using the Mel-spectrogram, at 93.60%, whereas the best accuracy obtained by the model using ANNs with a hierarchical structure was 73.40%, as shown in Table 4.6.

Table 5.4: Performance comparison of four audio feature image representations on the MUSDB18 dataset.

| Index | Class | Mel-spectrogram | | | dB-MFCCs | | | CQT | | | CENS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| 0 | vocals | **1.0000** | **1.0000** | **1.0000** | 0.9655 | 0.9655 | 0.9655 | **1.0000** | 0.9655 | 0.9825 | 0.5000 | 0.5172 | 0.5085 |
| 1 | drums | 0.8710 | **0.9310** | **0.9000** | **0.8889** | 0.8276 | 0.8571 | 0.8710 | **0.9310** | **0.9000** | 0.4667 | 0.7241 | 0.5676 |
| 2 | bass | **1.0000** | 0.9655 | 0.9825 | **1.0000** | 0.8966 | 0.9455 | **1.0000** | **1.0000** | **1.0000** | 0.6667 | 0.5517 | 0.6038 |
| 3 | vocals_drums | **1.0000** | **0.8966** | **0.9455** | 0.8846 | 0.7931 | 0.8364 | 0.9286 | **0.8966** | 0.9123 | 0.2941 | 0.1724 | 0.2174 |
| 4 | vocals_bass | **1.0000** | **0.9310** | **0.9643** | 0.8667 | 0.8966 | 0.8814 | 0.9630 | 0.8966 | 0.9286 | 0.4103 | 0.5517 | 0.4706 |
| 5 | drums_bass | 0.8182 | 0.9310 | 0.8710 | 0.7778 | **0.9655** | 0.8615 | **0.9032** | **0.9655** | **0.9333** | 0.2727 | 0.2069 | 0.2353 |
| 6 | vocals_drums_bass | **0.8966** | **0.8966** | **0.8966** | 0.7931 | 0.7931 | 0.7931 | 0.8621 | 0.8621 | 0.8621 | 0.5000 | 0.4483 | 0.4727 |
| | **Accuracy** | | **0.9360** | | | 0.8768 | | | 0.9310 | | | 0.4532 | |




(a) Mel-spectrogram (b) db-MFCCs

(c) CQT (d) CENS

Figure 5.6: Confusion matrices for the models using CNNs with four features on the MUSDB18 dataset.

Unexpectedly, the model using the CQT achieved an accuracy of 93.10%, which was very close to the model using the Mel-spectrogram, even though the CQT images were only one-third the size of the Mel-spectrogram images.

For the models using the dB-MFCCs and CENS, the performance of the model using

the dB-MFCCs can be regarded as reasonable, given the smaller number of its input images, while the low performance of the model using the CENS was what we expected. Because the CENS is specialized for musical attributes such as chord progression, it is not well-suited for musical instrument classification.

Across all models, a small portion of mixed-instrument samples are misclassified as the *drums* class, which may reflect a masking effect between the *drums* class and the other classes.

## 5.3 Empirical Experiments on the MedleyDB Dataset

In this section, we present the experiments conducted on the MedleyDB dataset. We used the same dataset as described in Section 4.3.1, and the model architecture and data preprocessing procedures were the same as those in Sections 5.1.1 and 5.1.2, respectively.

We first present the same experiment as in Section 5.2.1 using the MedleyDB dataset, followed by an experiment examining how performance changes with and without Gaussian noise injection in the dataset. Finally, we present experiments using multiple audio feature images in a model.

### 5.3.1 Experiments on the MedleyDB Dataset with Audio Feature Images

For the experiments on the MedleyDB dataset, we used three subsets of the dataset, rather than two as in the MUSDB18 experiments, to account for the larger number of classes. Our goal was to examine how the model's performance changed as the number of classes increased.

In the first experiment, we used only pure-instrument classes (7 classes); in the second, we additionally included mixed 2-3-instrument classes (13 classes in total); and in the third, we included all remaining classes (18 classes in total). Table 5.5 shows the overall performance across these experiments.

Table 5.5: Performance comparison of four audio feature image representations on the MedleyDB dataset.

| Features | Shapes | Pure | 1–3 Insts. | All |
|---|---|---|---|---|
| Mel-spectrogram | (256, 5168, 2) | **85.71**% | **70.38**% | **54.22**% |
| dB-MFCCs | (40, 5168, 2) | 83.71% | 64.08% | 47.00% |
| CQT | (84, 5168, 2) | 78.71% | 63.15% | 50.44% |
| CENS | (12, 5168, 2) | 56.43% | 30.08% | 21.44% |

### Classifying Pure-instrument Classes

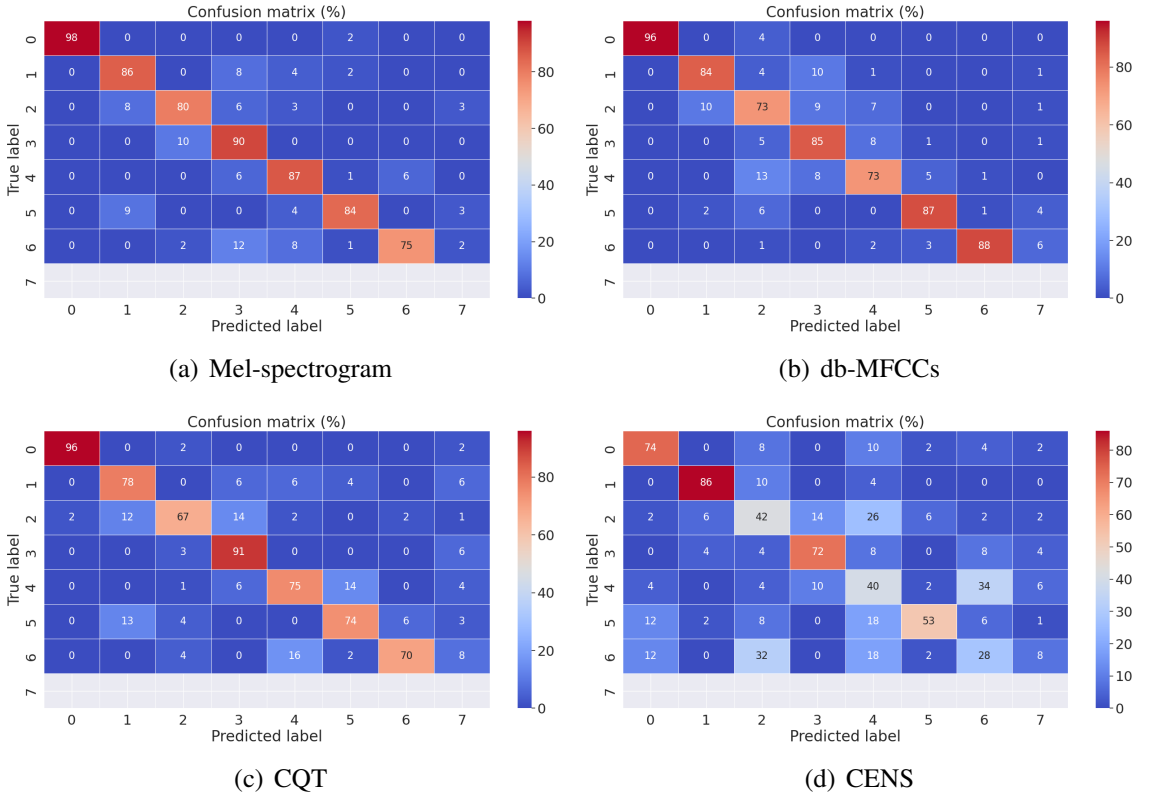
This experiment was conducted using pure-instrument classes. Table 5.6 and Figure 5.7 show the results and the confusion matrices for each model, respectively.

Table 5.6: Performance comparison for the pure instrument classes on the MedleyDB dataset.

| Index | Class | Mel-spectrogram | | | dB-MFCCs | | | CQT | | | CENS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| 0 | Bass | **1.0000** | **0.9800** | **0.9899** | **1.0000** | 0.9600 | 0.9796 | 0.9796 | 0.9600 | 0.9697 | 0.7115 | 0.7400 | 0.7255 |
| 1 | Drums | 0.8350 | 0.8600 | 0.8473 | 0.8750 | 0.8400 | 0.8571 | 0.7573 | 0.7800 | 0.7685 | **0.8776** | **0.8600** | **0.8687** |
| 2 | Guitars | **0.8696** | **0.8000** | **0.8333** | 0.6887 | 0.7300 | 0.7087 | 0.8272 | 0.6700 | 0.7403 | 0.3889 | 0.4200 | 0.4038 |
| 3 | Piano | 0.7377 | 0.9000 | 0.8108 | 0.7589 | 0.8500 | 0.8019 | **0.7778** | **0.9100** | **0.8387** | 0.7500 | 0.7200 | 0.7347 |
| 4 | Strings | **0.8208** | **0.8700** | **0.8447** | 0.8022 | 0.7300 | 0.7644 | 0.7576 | 0.7500 | 0.7538 | 0.3226 | 0.4000 | 0.3571 |
| 5 | Vocals | **0.9333** | 0.8400 | 0.8842 | 0.9062 | **0.8700** | **0.8878** | 0.7872 | 0.7400 | 0.7629 | 0.8154 | 0.5300 | 0.6424 |
| 6 | Woodwinds | 0.9259 | 0.7500 | 0.8287 | **0.9778** | **0.8800** | **0.9263** | 0.8974 | 0.7000 | 0.7865 | 0.3415 | 0.2800 | 0.3077 |
| **Accuracy** | | | **0.8571** | | | 0.8371 | | | 0.7871 | | | 0.5643 | |

The best accuracy was achieved by the model using the Mel-spectrogram, at 85.71%, whereas the best accuracy obtained by the model using ANNs with a hierarchical structure was 75.71%, as shown in Table 4.14. Like the experiments on the MUSDB18 dataset, the models using CNNs outperformed all models using ANNs except for the model using the CENS. However, unlike in the MUSDB18 experiments, where the best accuracy improved from 73.40% to 93.60%, the performance improvement on the MedleyDB dataset was not as much as the experiments on the MUSDB18. This is likely because the MedleyDB dataset is larger, including audio clips with Gaussian noise, and contains a greater number and variety of instruments.

The model using the Mel-spectrogram achieved the best performance because it provides more information about the audio signal and involves a larger image size than the




(a) Mel-spectrogram (b) db-MFCCs

(c) CQT (d) CENS

Figure 5.7: Confusion matrices for the pure instrument classes on the MedleyDB dataset.

other audio features. The performance of the model using the CENS was also the lowest, as expected, likely due to the smaller image size.

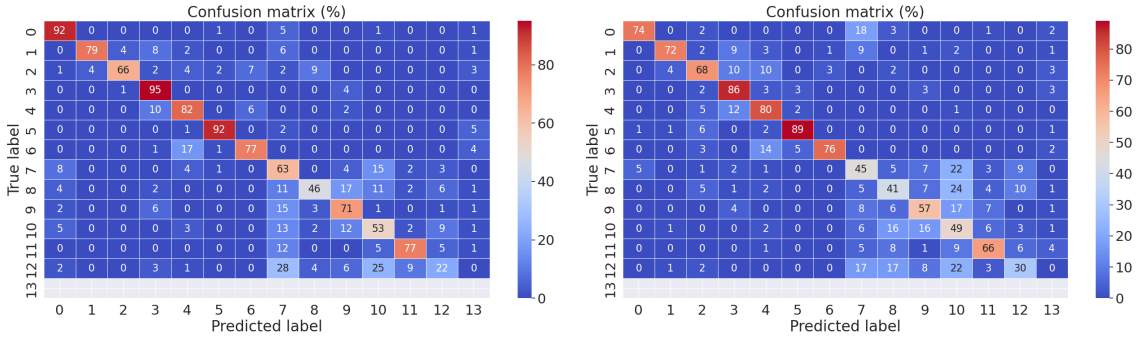
### Classifying One to Three Instrument Classes

Next, we conducted experiments on classes that contained between one and three instruments. Table 5.7 presents the experimental results, and Figure 5.8 shows the corresponding confusion matrices. As for the notation of classes, the initial letters of the seven (7) pure instrument classes in the MedleyDB dataset are all distinct. Therefore, the mixed-instrument classes are denoted using these initial letters. For example, the *b_d_s_v* class represents a mixed-instrument class consisting of *bass*, *drums*, *strings*, and *vocals*.

The performance decreased as the number of classes increased, and the mixed-instrument classes made the models more prone to confusion. The model using the Mel-spectrogram still achieved the best performance. The performance of the model using the CQT de-

Table 5.7: Performance comparison for one to three instrument classes on the MedleyDB dataset.

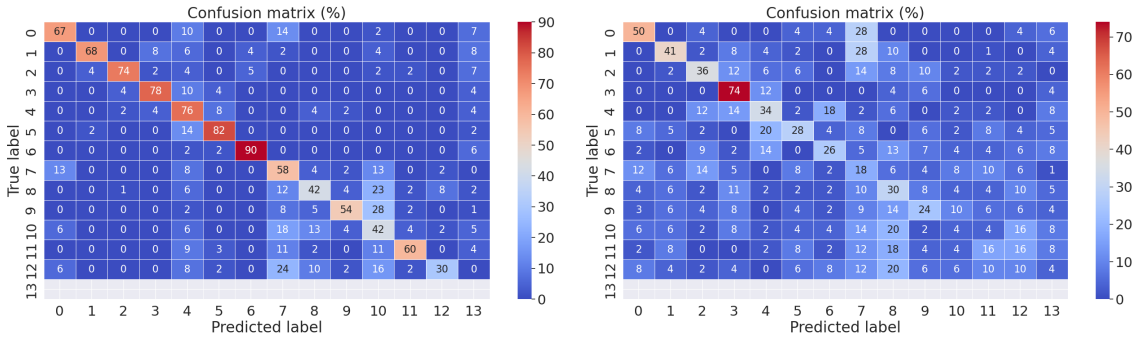
| Index | Class | Mel-spectrogram | | | dB-MFCCs | | | CQT | | | CENS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| 0 | Bass | 0.8070 | **0.9200** | **0.8598** | **0.9250** | 0.7400 | 0.8222 | 0.7283 | 0.6700 | 0.6979 | 0.5263 | 0.5000 | 0.5128 |
| 1 | Drums | **0.9518** | **0.7900** | **0.8634** | 0.9114 | 0.7200 | 0.8045 | 0.9189 | 0.6800 | 0.7816 | 0.4881 | 0.4100 | 0.4457 |
| 2 | Guitars | **0.9296** | 0.6600 | 0.7719 | 0.7083 | 0.6800 | 0.6939 | 0.9136 | **0.7400** | **0.8177** | 0.4045 | 0.3600 | 0.3810 |
| 3 | Piano | 0.7480 | **0.9500** | **0.8370** | 0.6935 | 0.8600 | 0.7679 | **0.8478** | 0.7800 | 0.8125 | 0.5068 | 0.7400 | 0.6016 |
| 4 | Strings | **0.7193** | **0.8200** | **0.7664** | 0.6780 | 0.8000 | 0.7339 | 0.4720 | 0.7600 | 0.5824 | 0.3542 | 0.3400 | 0.3469 |
| 5 | Vocals | **0.9485** | **0.9200** | **0.9340** | 0.8990 | 0.8900 | 0.8945 | 0.8119 | 0.8200 | 0.8159 | 0.3784 | 0.2800 | 0.3218 |
| 6 | Woodwinds | 0.8556 | 0.7700 | 0.8105 | **0.9500** | 0.7600 | 0.8444 | 0.9091 | **0.9000** | **0.9045** | 0.3611 | 0.2600 | 0.3023 |
| 7 | b_d | **0.4013** | **0.6300** | **0.4903** | 0.3982 | 0.4500 | 0.4225 | 0.3946 | 0.5800 | 0.4696 | 0.1125 | 0.1800 | 0.1385 |
| 8 | b_d_g | **0.7188** | **0.4600** | **0.5610** | 0.4184 | 0.4100 | 0.4141 | 0.5250 | 0.4200 | 0.4667 | 0.2013 | 0.3000 | 0.2410 |
| 9 | b_d_p | 0.6121 | 0.7100 | **0.6574** | 0.5700 | 0.5700 | 0.5700 | **0.7941** | 0.5400 | 0.6429 | 0.3117 | 0.2400 | 0.2712 |
| 10 | b_d_s | **0.4775** | **0.5300** | **0.5024** | 0.3356 | 0.4900 | 0.3984 | 0.2979 | 0.4200 | 0.3485 | 0.0870 | 0.0400 | 0.0548 |
| 11 | b_d_v | **0.8370** | **0.7700** | **0.8021** | 0.7333 | 0.6600 | 0.6947 | 0.8333 | 0.6000 | 0.6977 | 0.2388 | 0.1600 | 0.1916 |
| 12 | b_d_w | 0.4783 | 0.2200 | 0.3014 | 0.5172 | **0.3000** | 0.3797 | **0.7143** | **0.3000** | **0.4225** | 0.1250 | 0.1000 | 0.1111 |
| **Accuracy** | | **0.7038** | | | 0.6408 | | | 0.6315 | | | 0.3008 | | |




(a) Mel-spectrogram




(b) db-MFCCs




(c) CQT




(d) CENS

Figure 5.8: Confusion matrices for 1-3-instrument classes on the MedleyDB dataset.

creased moderately, from 78.71% to 63.15%, while that of the model using the dB-MFCC decreased to a slightly greater amount, from 83.71% to 64.08%, and the CENS showed a

further decrease.

According to the confusion matrices, the pure-instrument classes were classified well. However, starting from the *bass_drums* class, the performances dropped noticeably.

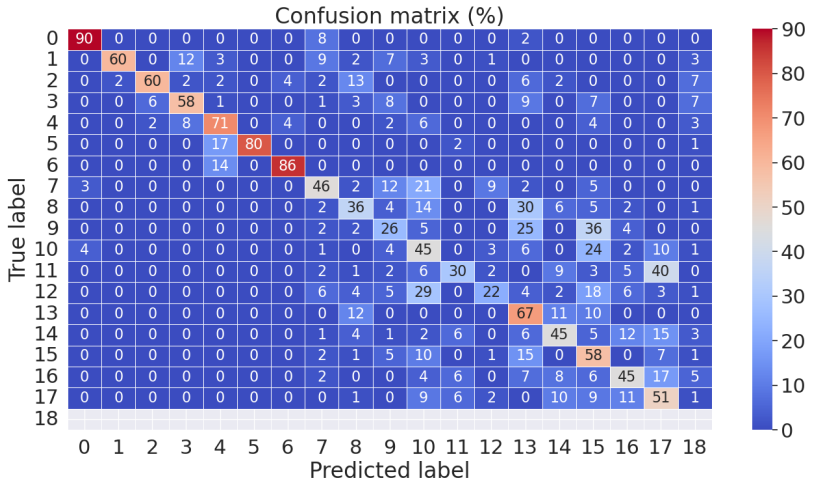
### Classifying All Classes

Based on the previous experiments, we then evaluated the models using all classes. Table 5.8 presents the results for all classes, and Figure 5.9 shows the confusion matrices of each model.

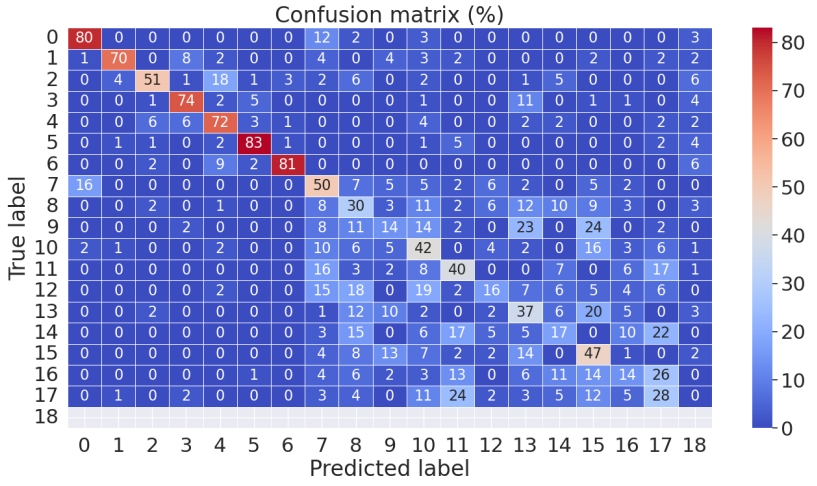
Table 5.8: Performance comparison for all classes on the MedleyDB dataset.

| Index | Class | Mel-spectrogram | | | dB-MFCCs | | | CQT | | | CENS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| 0 | Bass | **0.9278** | **0.9000** | **0.9137** | 0.8081 | 0.8000 | 0.8040 | 0.7957 | 0.7400 | 0.7668 | 0.5565 | 0.6400 | 0.5953 |
| 1 | Drums | **0.9677** | 0.6000 | 0.7407 | 0.9091 | 0.7000 | **0.7910** | 0.8506 | **0.7400** | **0.7914** | 0.5625 | 0.3600 | 0.4390 |
| 2 | Guitars | **0.8824** | 0.6000 | **0.7143** | 0.7846 | 0.5100 | 0.6182 | 0.7079 | **0.6300** | 0.6667 | 0.3846 | 0.2000 | 0.2632 |
| 3 | Piano | 0.7250 | 0.5800 | 0.6444 | 0.7957 | 0.7400 | 0.7668 | **0.8077** | **0.8400** | **0.8235** | 0.6139 | 0.6200 | 0.6169 |
| 4 | Strings | **0.6574** | 0.7100 | 0.6827 | 0.6545 | 0.7200 | **0.6857** | 0.5827 | **0.7400** | 0.6520 | 0.4130 | 0.3800 | 0.3958 |
| 5 | Vocals | **1.0000** | **0.8000** | **0.8889** | 0.8737 | **0.8300** | 0.8513 | 0.8642 | 0.7000 | 0.7735 | 0.4444 | 0.4400 | 0.4422 |
| 6 | Woodwinds | 0.9149 | **0.8600** | **0.8866** | 0.9419 | 0.8100 | 0.8710 | **0.9494** | 0.7500 | 0.8380 | 0.2727 | 0.1200 | 0.1667 |
| 7 | b_d | **0.5476** | 0.4600 | **0.5000** | 0.3571 | 0.5000 | 0.4167 | 0.3420 | **0.6600** | 0.4505 | 0.0818 | 0.0900 | 0.0857 |
| 8 | b_d_g | **0.4444** | 0.3600 | **0.3978** | 0.2344 | 0.3000 | 0.2632 | 0.2692 | **0.4200** | 0.3281 | 0.2174 | 0.1000 | 0.1370 |
| 9 | b_d_p | **0.3421** | 0.2600 | 0.2955 | 0.2414 | 0.1400 | 0.1772 | 0.3084 | **0.3300** | **0.3188** | 0.0833 | 0.0600 | 0.0698 |
| 10 | b_d_s | 0.2922 | **0.4500** | **0.3543** | **0.2958** | 0.4200 | 0.3471 | 0.2661 | 0.2900 | 0.2775 | 0.0781 | 0.0500 | 0.0610 |
| 11 | b_d_v | **0.6000** | 0.3000 | 0.4000 | 0.3604 | 0.4000 | 0.3791 | 0.4486 | **0.4800** | **0.4638** | 0.1250 | 0.2000 | 0.1538 |
| 12 | b_d_w | 0.5500 | 0.2200 | 0.3143 | 0.3721 | 0.1600 | 0.2238 | **0.5581** | **0.2400** | **0.3357** | 0.0612 | 0.0600 | 0.0606 |
| 13 | b_d_g_p | **0.3743** | **0.6700** | **0.4803** | 0.2960 | 0.3700 | 0.3289 | 0.3154 | 0.4700 | 0.3775 | 0.1360 | 0.1700 | 0.1511 |
| 14 | b_d_g_v | **0.4839** | **0.4500** | **0.4663** | 0.2464 | 0.1700 | 0.2012 | 0.3529 | 0.3000 | 0.3243 | 0.1111 | 0.1700 | 0.1344 |
| 15 | b_d_p_s | 0.3053 | **0.5800** | **0.4000** | 0.3032 | 0.4700 | 0.3686 | **0.3429** | 0.2400 | 0.2824 | 0.1600 | 0.0800 | 0.1067 |
| 16 | b_d_p_v | 0.5172 | **0.4500** | **0.4813** | 0.2593 | 0.1400 | 0.1818 | **0.5714** | 0.4000 | 0.4706 | 0.0536 | 0.0600 | 0.0566 |
| 17 | b_d_s_v | **0.3566** | **0.5100** | **0.4198** | 0.2478 | 0.2800 | 0.2629 | 0.3438 | 0.1100 | 0.1667 | 0.0577 | 0.0600 | 0.0588 |
| **Accuracy** | | | **0.5422** | | | 0.4700 | | | 0.5044 | | | 0.2144 | |

The performance of the model for all classes exceeded that of the previous model introduced in Section 4.3.2, improving from 34.56% to 54.22%. The model using the Mel-spectrogram outperformed the others on the mixed four-instrument classes. However, its performance on the pure-instrument classes and on the mixed classes of 2 to 3 instruments decreased compared to the previous experiments. In contrast, the model using the CQT showed better results on the pure-instrument classes and on the mixed classes of 2 to 3 instruments, and it eventually exceeded the overall performance of the dB-MFCCs. Never-




(a) Mel-spectrogram

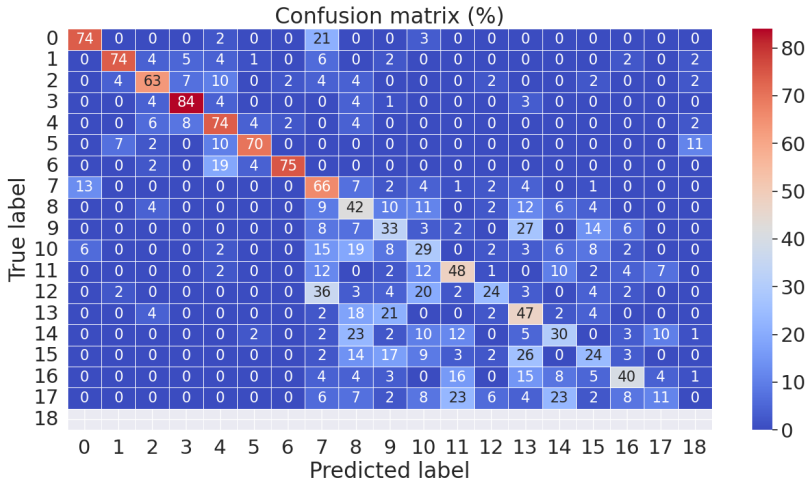



(b) db-MFCCs

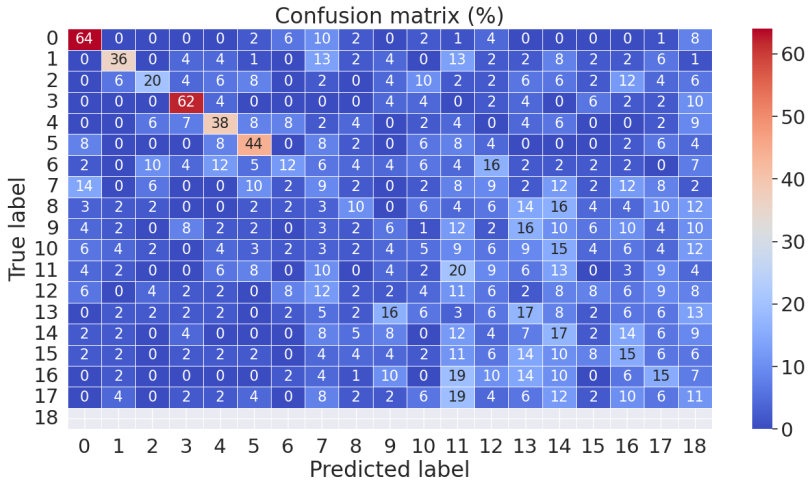
Figure 5.9: Confusion matrices for all classes on the MedleyDB dataset.

theless, the overall performance of the model using the Mel-spectrogram remained the best, with higher performance for the classes that contained more mixed instruments.

Classifying mixed-instrument classes in our derived dataset from MedleyDB may be more challenging than classifying mixed-instrument classes with the same number of instruments in other datasets. One reason may be that these classes are based on the *bass* and *drums* classes. As a result, adding one or two additional instruments may provide only subtle differences, making the classes harder for the models to distinguish.




(c) CQT




(d) CENS

Figure 5.9: Confusion matrices for all classes on the MedleyDB dataset (continued).

### 5.3.2 Performance Comparison With and Without Gaussian Noise Injection on the MedleyDB Dataset

Since we used Gaussian-noise-injected audio clips for upsampling, we sought to investigate the effect of Gaussian noise on the model. For this purpose, we constructed a separate dataset specifically for this experiment.

We first collected pure-instrument audio clips without Gaussian noise for each class from the training and test sets we used in the previous experiments. Because the *woodwinds* class had the fewest audio clips before the upsampling stage, the number of clips in the other

classes was matched to the *woodwinds* class. As a result, each class contained 126 audio clips in the training set and 32 audio clips in the test set.

After that, we injected Gaussian noise into each class and generated mixed-instrument classes for each set, resulting in two separate datasets: one with Gaussian noise and one without Gaussian noise. We then conducted experiments on these two datasets. Table 5.9 presents an overview of the results of these experiments, and Table 5.10 presents a detailed comparison for all classes.

Table 5.9: Effect of Gaussian noise on classification accuracy.

| **Features** | **Shapes** | **Without Gaussian noise** | | | **With Gaussian noise** | | |
|---|---|---|---|---|---|---|---|
| | | **Pure** | **1–3 Insts.** | **All** | **Pure** | **1–3 Insts.** | **All** |
| **Mel-spectrogram** | (256, 5168, 2) | **78.12%** | **65.14%** | **52.08%** | 75.00% | 64.18% | 51.39% |
| **dB-MFCCs** | (40, 5168, 2) | 76.34% | **58.41%** | **43.40%** | **78.12%** | 57.69% | 41.67% |
| **CQT** | (84, 5168, 2) | **75.89%** | **61.54%** | **44.79%** | 71.43% | 57.21% | 42.36% |
| **CENS** | (12, 5168, 2) | 54.46% | 25.24% | **21.35%** | **56.25%** | **25.48%** | 21.01% |

Table 5.10: Effect of Gaussian noise on classification accuracy across all classes.

| **Index** | **Class** | **Without Gaussian Noise** | | | | **With Gaussian Noise** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **Mel** | **MFCCs** | **CQT** | **CENS** | **Mel** | **MFCCs** | **CQT** | **CENS** |
| 0 | bass | **93.75%** | 87.50% | 87.50% | 50.00% | 78.12% | 62.50% | 68.75% | 62.50% |
| 1 | drums | **87.50%** | 53.12% | 78.12% | 53.12% | 68.75% | 71.88% | 71.88% | 43.75% |
| 2 | guitars | 75.00% | 53.12% | 50.00% | 25.00% | **78.12%** | 59.38% | 68.75% | 12.50% |
| 3 | piano | **93.75%** | 78.12% | 87.50% | 75.00% | 75.00% | 78.12% | 81.25% | 62.50% |
| 4 | strings | **68.75%** | 56.25% | 65.62% | 31.25% | 50.00% | 53.12% | 59.38% | **68.75%** |
| 5 | vocals | 71.88% | 65.62% | 68.75% | 12.50% | 68.75% | **81.25%** | **81.25%** | 31.25% |
| 6 | woodwinds | 62.50% | **84.38%** | 28.12% | 31.25% | 68.75% | 56.25% | 53.12% | 25.00% |
| 7 | b_d | **65.62%** | 34.38% | 46.88% | 12.50% | 56.25% | 37.50% | 62.50% | 25.00% |
| 8 | b_d_g | 31.25% | 31.25% | 43.75% | 9.38% | **50.00%** | 31.25% | 18.75% | 0.00% |
| 9 | b_d_p | 6.25% | 15.62% | **50.00%** | 25.00% | 43.75% | 21.88% | 12.50% | 18.75% |
| 10 | b_d_s | 46.88% | 28.12% | 25.00% | 0.00% | 28.12% | 31.25% | **50.00%** | 0.00% |
| 11 | b_d_v | **53.12%** | 21.88% | 50.00% | 21.88% | 50.00% | 28.12% | 21.88% | 0.00% |
| 12 | b_d_w | 34.38% | 21.88% | 25.00% | 0.00% | **43.75%** | 31.25% | 15.62% | 6.25% |
| 13 | b_d_g_p | 46.88% | **59.38%** | 21.88% | 12.50% | 50.00% | 31.25% | 6.25% | 6.25% |
| 14 | b_d_g_v | 31.25% | **50.00%** | 37.50% | 12.50% | 28.12% | 6.25% | 6.25% | 6.25% |
| 15 | b_d_p_s | 21.88% | 6.25% | 12.50% | 6.25% | 28.12% | **31.25%** | 25.00% | 0.00% |
| 16 | b_d_p_v | 18.75% | 18.75% | 15.62% | 6.25% | 12.50% | 18.75% | **21.88%** | 9.38% |
| 17 | b_d_s_v | 28.12% | 15.62% | 12.50% | 0.00% | **46.88%** | 18.75% | 37.50% | 0.00% |
| **Accuracy** | | **52.08%** | 43.40% | 44.79% | 21.35% | 51.39% | 41.67% | 42.36% | 21.01% |

The results of the experiments show that Gaussian noise slightly degrades the overall performance of the models. As mentioned in Section 3.2.2, Gaussian noise was injected

with the SNR range set between 20 and 35 dB. We did this to increase the number of audio clips and to address class imbalance. Because Gaussian noise does not consist of stable and sustained frequency components and randomly changes its frequencies at every time, this property may degrade the model's performance. However, it appears to be a little beneficial for some mixed-instrument classes, which outperformed the model without Gaussian noise. This setting could also be further investigated as a way to improve model performance, for example, by incorporating it into the training pipeline as a targeted reinforcement for specific classes, similar to how dithering noise is used when reducing the bit depth of digital audio or images.

### 5.3.3 Experiments on the MedleyDB Dataset with Combined Audio Feature Images

While exploring the audio feature extraction stage, we found that Mel-spectrograms, MFCCs, and CQT could be extracted in the same shapes. So we applied the same processing as in Section 5.3.1, but set the number of features for each audio representation to 80. As a result, all of them had the same shape of $80 \times 5168 \times 2$.

Sahoo et al. [51] conducted a similar study using CQT, Mel-spectrogram, and semitone spectrogram. But their input format was a three-channel 2D image. However, in our study, we utilized stereo channels and applied different combinations of audio features. When two features were combined, the input shape changed to $80 \times 5168 \times 4$, corresponding to four channels, and when three features were combined, their shape became $80 \times 5168 \times 6$, corresponding to six channels. Table 5.11 shows an overview of the results across feature combinations, and Table 5.12 presents detailed results for each class across all feature combinations.

The results were interesting. The combination of the Mel-spectrogram and the dB-MFCCs showed the best performance. The previous best results were those presented in Table 5.5. The accuracy for the pure-instrument classes increased from 85.71% to 86.86%, and the accuracy for all classes increased from 54.22% to 54.61%. However, for the

Table 5.11: Classification accuracy comparison across single and combined features

| Features | Shapes | Pure | 1–3 Insts. | All |
|---|---|---|---|---|
| Mel-spectrogram | (80, 5168, 2) | 83.29% | 68.62% | 49.39% |
| dB-MFCCs | (80, 5168, 2) | 83.43% | 67.00% | 50.56% |
| CQT | (80, 5168, 2) | 78.57% | 63.08% | 50.17% |
| Mel_dB-MFCCs | (80, 5168, 4) | **86.86%** | 68.46% | **54.61%** |
| Mel_CQT | (80, 5168, 4) | 85.14% | **69.62%** | 53.78% |
| dB-MFCCs_CQT | (80, 5168, 4) | 82.71% | 66.08% | 51.28% |
| Mel_dB-MFCCs_CQT | (80, 5168, 6) | 86.29% | 67.92% | 51.83% |

Table 5.12: Classification accuracy for all classes and feature combinations

| Index | Class | Single Features | | | Dual Fusion | | | Triple |
|---|---|---|---|---|---|---|---|---|
| | | Mel | MFCC | CQT | M+MF | M+C | MF+C | All |
| 0 | bass | 73.00% | **90.00%** | 80.00% | 76.00% | 85.00% | 88.00% | 75.00% |
| 1 | drums | 64.00% | 68.00% | 65.00% | 71.00% | **76.00%** | 74.00% | 73.00% |
| 2 | guitars | 67.00% | 54.00% | 63.00% | **73.00%** | **73.00%** | 51.00% | 63.00% |
| 3 | piano | 74.00% | 77.00% | 84.00% | 87.00% | 80.00% | **89.00%** | 80.00% |
| 4 | strings | 70.00% | 72.00% | 82.00% | **83.00%** | 78.00% | 78.00% | 80.00% |
| 5 | vocals | 79.00% | 81.00% | 82.00% | 75.00% | **87.00%** | 76.00% | **87.00%** |
| 6 | woodwinds | 57.00% | 84.00% | 73.00% | **95.00%** | 75.00% | 82.00% | 84.00% |
| 7 | b_d | **63.00%** | 52.00% | 51.00% | **63.00%** | 44.00% | 43.00% | 40.00% |
| 8 | b_d_g | 14.00% | 32.00% | **40.00%** | 31.00% | 37.00% | 34.00% | 27.00% |
| 9 | b_d_p | 47.00% | **52.00%** | 29.00% | 51.00% | 38.00% | 45.00% | 36.00% |
| 10 | b_d_s | 32.00% | 32.00% | **46.00%** | 39.00% | **46.00%** | 45.00% | 45.00% |
| 11 | b_d_v | 39.00% | 33.00% | 47.00% | 47.00% | 52.00% | **53.00%** | 49.00% |
| 12 | b_d_w | **29.00%** | 20.00% | 23.00% | 22.00% | 24.00% | 24.00% | 24.00% |
| 13 | b_d_g_p | 26.00% | 28.00% | 41.00% | **44.00%** | 34.00% | 26.00% | 35.00% |
| 14 | b_d_g_v | 34.00% | 32.00% | 31.00% | 30.00% | 32.00% | 30.00% | **41.00%** |
| 15 | b_d_p_s | 36.00% | 45.00% | 22.00% | 44.00% | **46.00%** | 38.00% | 42.00% |
| 16 | b_d_p_v | **48.00%** | 41.00% | 16.00% | 33.00% | 39.00% | 26.00% | 23.00% |
| 17 | b_d_s_v | 37.00% | 17.00% | 28.00% | 18.00% | 22.00% | 21.00% | 29.00% |
| **Overall Accuracy** | | 49.39% | 50.56% | 50.17% | **54.61%** | 53.78% | 51.28% | 51.83% |

mixed-instrument classes with one to three instruments, the Mel-spectrogram alone showed slightly better performance, with an accuracy of 70.38%, whereas the best result from this experiment for those classes was obtained with the combination of the Mel-spectrogram and the CQT, with an accuracy of 69.62%.

According to Table 5.12, the combinations of the Mel-spectrogram with the dB-MFCCs and with the CQT generally achieved better performance than the others, whereas the combination of the dB-MFCCs and the CQT performed worse than the other two-feature combinations. These results suggest that the dB-MFCCs and the CQT provide complementary

information that is not captured by the Mel-spectrogram, while the Mel-spectrogram itself contains the most critical information that is not present in the dB-MFCCs or the CQT.

One interesting observation is that the combination of all three features did not yield the best performance. It outperformed single-feature configurations but achieved slightly lower accuracy than most two-feature combinations, except for the combination of the dB-MFCCs and the CQT. This suggests that the dB-MFCCs and the CQT may share redundant information and are less complementary when used together.

From the perspective of data size, the combination of the Mel-spectrogram and the dB-MFCCs is preferable to the single Mel-spectrogram, since it reduces the input size from $256 \times 5168 \times 2$ = 2,646,016 to $80 \times 5168 \times 4$ = 1,653,760.

**Softmax-based Experiments with Combined Audio Feature Images**

We have presented experimental results demonstrating that combining multiple audio features generally yields better performance than using a single feature. However, we still faced the issue that some audio clips were predicted as the *others* class, which is described in Section 4.3.1.

To address this issue, we employed the Softmax activation function with label encoding instead of the Sigmoid activation function with one-hot encoding. Although we could have reduced the number of classes to a power of two, we chose not to do so because that would not be directly comparable to the previous experiments. Table 5.13 shows a summary of the results, and Table 5.14 presents detailed results for each class across all feature combinations.

All models showed slightly better performances than the models using the Sigmoid activation function with one-hot encoding. The accuracy for the pure-instrument classes increased from 85.71% to 87.71%, for the mixed-instrument classes with one to three instruments from 69.62% to 70.54%, and for all classes from 54.61% to 54.72%.

The combination of the Mel-spectrogram and the dB-MFCCs remained the best per-

Table 5.13: Classification accuracy comparison across single and combined features using Softmax

| Features | Shapes | Pure | 1–3 Insts. | All |
|---|---|---|---|---|
| Mel-spectrogram | (80, 5168, 2) | 87.43% | 69.46% | 53.17% |
| dB-MFCCs | (80, 5168, 2) | 85.29% | 67.23% | 52.22% |
| CQT | (80, 5168, 2) | 79.57% | 65.85% | 50.33% |
| Mel_dB-MFCCs | (80, 5168, 4) | 87.43% | **70.54%** | **54.72%** |
| Mel_CQT | (80, 5168, 4) | **87.71%** | 69.92% | 53.94% |
| dB-MFCCs_CQT | (80, 5168, 4) | 83.43% | 67.46% | 52.00% |
| Mel_dB-MFCCs_CQT | (80, 5168, 6) | 86.57% | 69.77% | 52.56% |

Table 5.14: Classification accuracy per class and feature combination using Softmax

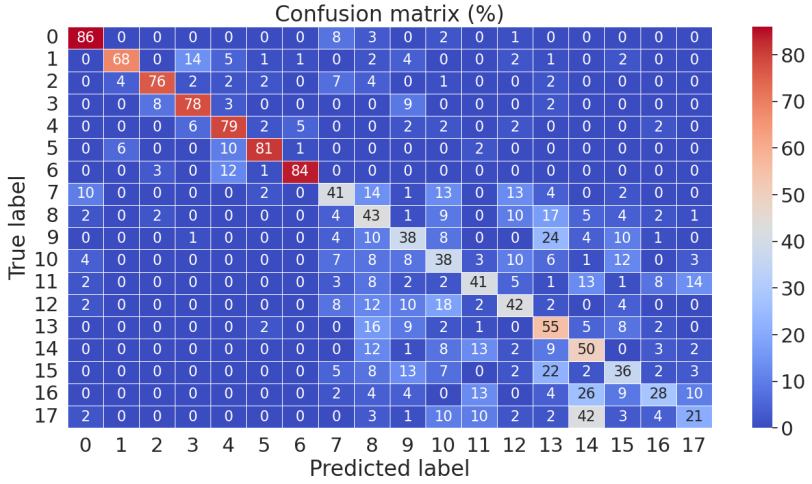
| Idx | Class | Single Features | | | Dual Fusion | | | Triple |
|---|---|---|---|---|---|---|---|---|
| | | Mel | MFCC | CQT | M+MF | M+C | MF+C | All |
| 0 | bass | 75.00% | 70.00% | **87.00%** | 86.00% | 87.00% | 74.00% | 81.00% |
| 1 | drums | 74.00% | 74.00% | 56.00% | 68.00% | 77.00% | 72.00% | **86.00%** |
| 2 | guitars | 60.00% | 68.00% | 62.00% | **76.00%** | 70.00% | 67.00% | 65.00% |
| 3 | piano | **94.00%** | 93.00% | 83.00% | 78.00% | 85.00% | 74.00% | 83.00% |
| 4 | strings | 74.00% | 69.00% | 88.00% | 79.00% | **89.00%** | 74.00% | 75.00% |
| 5 | vocals | 80.00% | 80.00% | 82.00% | 81.00% | 82.00% | **93.00%** | 88.00% |
| 6 | woodwinds | 68.00% | 87.00% | 87.00% | 84.00% | 72.00% | 75.00% | **97.00%** |
| 7 | b_d | 57.00% | 50.00% | 37.00% | 41.00% | 36.00% | **58.00%** | 47.00% |
| 8 | b_d_g | 21.00% | 26.00% | 35.00% | **43.00%** | 31.00% | 28.00% | 22.00% |
| 9 | b_d_p | 38.00% | 39.00% | 34.00% | 38.00% | **43.00%** | 37.00% | 33.00% |
| 10 | b_d_s | 40.00% | **55.00%** | 36.00% | 38.00% | 34.00% | 31.00% | 43.00% |
| 11 | b_d_v | 49.00% | 24.00% | 28.00% | 41.00% | 41.00% | 44.00% | **53.00%** |
| 12 | b_d_w | 23.00% | 25.00% | 22.00% | **42.00%** | 15.00% | 14.00% | 37.00% |
| 13 | b_d_g_p | 40.00% | 37.00% | 33.00% | **55.00%** | 40.00% | 49.00% | 22.00% |
| 14 | b_d_g_v | 20.00% | 39.00% | 41.00% | **50.00%** | 41.00% | 38.00% | 22.00% |
| 15 | b_d_p_s | 48.00% | 39.00% | 37.00% | 36.00% | 42.00% | 37.00% | **63.00%** |
| 16 | b_d_p_v | **44.00%** | 39.00% | 33.00% | 28.00% | 39.00% | 42.00% | 29.00% |
| 17 | b_d_s_v | **52.00%** | 26.00% | 25.00% | 21.00% | 47.00% | 29.00% | 33.00% |
| **Overall Accuracy** | | 53.17% | 52.22% | 50.33% | **54.72%** | 53.94% | 52.00% | 52.56% |

forming feature combination, and the results indicate that the mixed-instrument classes were classified more accurately with combined features than in the previous experiments. The accuracy of the model using the Mel-spectrogram increased the most among all feature combinations compared to the previous experiments.

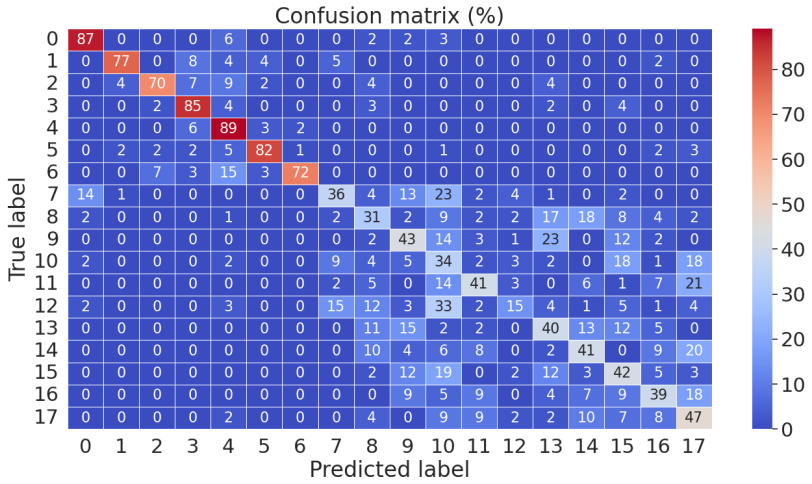
### Confusion Matrices of the Feature Combinations with the Highest Accuracy

In this section, we present the confusion matrices of the feature combinations that achieved the highest accuracy on the MedleyDB dataset, as shown in Figure 5.10. This

figure can be directly compared with Figure 5.9 in Section 5.3.1.




(a) Mel_dB-MFCCs

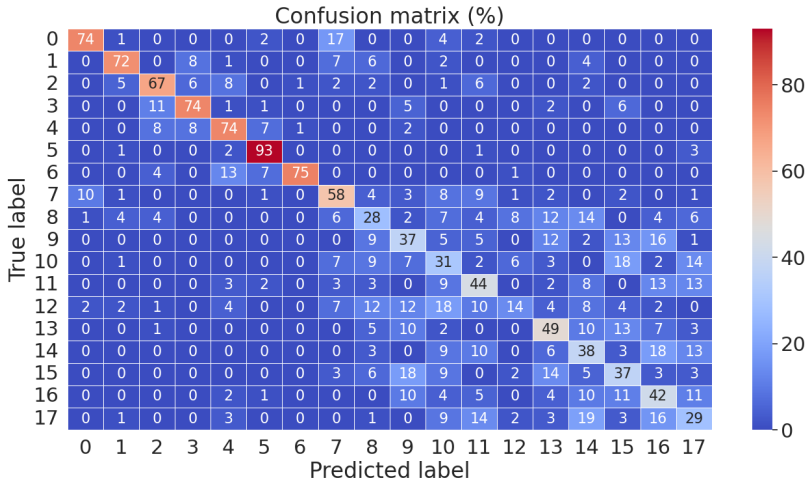



(b) Mel_CQT

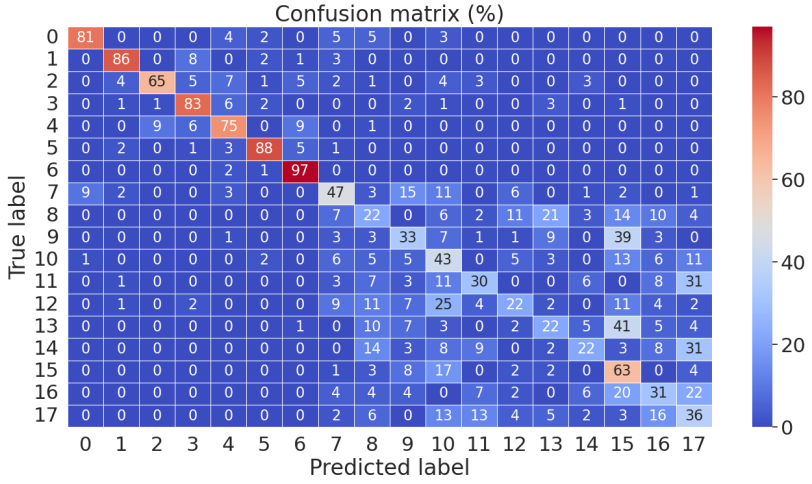
Figure 5.10: Confusion matrices of the feature combinations with the highest accuracy on the MedleyDB dataset.

These confusion matrices also suggest that combinations of audio feature images are more effective for classifying mixed-instrument classes, even though their accuracies remain unsatisfactory. However, this may still be a reasonable outcome, given that the seven (7) classes in fact correspond to 18 distinct instrument stems.

The combination of the Mel-spectrogram and the CQT tended to classify some audio clips in the *b_d_w* class as the *b_d_s* class. This may be because some woodwind instruments




(c) dB-MFCCs_CQT




(d) ALL

Figure 5.10: Confusion matrices of the feature combinations with the highest accuracy on the MedleyDB dataset (continued).

sound very similar to string instruments, especially in sustained articulations. The *b_d* class is in a similar situation; this may occur when pitched percussion instruments have pitches that are close to those of string instruments, given that the CQT representation emphasizes pitch information.

The combination of the Mel-spectrogram and the dB-MFCCs tended to classify some audio clips in the *b_d_s_v* class as the *b_d_g_v* class. This suggests that, when mixed with bass, drums, and vocals, string and guitar instruments may present similar characteristics,

leading to confusion between these two classes. The *b_d_p_v* class is in a similar situation, and these instruments may also be masked by the bass, drums, and vocals.

The combination of the Mel-spectrogram and the dB-MFCCs and the combination of the Mel-spectrogram and the CQT appear to provide more balanced performance than the other feature combinations, whereas the Mel-spectrogram alone in Figure 5.9 yields better performance for mixed 4-instrument classes. This observation suggests that combining these CNNs could achieve better overall performance.

## 5.4 Summary

This chapter presented experiments and evaluations of the models using CNNs on both the MUSDB18 and MedleyDB datasets. We first conducted experiments on the MUSDB18 dataset and then performed further experiments on the MedleyDB dataset, following the same procedure as in Chapter 4.

We first described the model architecture in detail. The network stacks convolutional layers with padding to preserve the same shape and applies max-pooling between them to halve the image size. Each convolutional layer uses the ReLU activation function, and the output layer uses a Sigmoid activation function.

We then presented detailed information about the audio feature images, along with example images. We obtained images of the Mel-spectrogram, MFCCs, CQT, and CENS. Each converted audio feature image has a different height (number of features), while they share the same width (the length of the audio clip) and depth (stereo channels).

We also introduced our proposed dB-MFCCs obtained by rescaling MFCCs on a logarithmic scale, which enhances weaker values and compresses stronger values. In our small experiments, the dB-MFCCs achieved higher accuracy (87.68%) than the original MFCCs (85.22%), and the same transformation was also applied to the CQT and the CENS.

Training CNNs took much longer than training ANNs. It required approximately one to three hours for the MUSDB18 dataset and three to more than ten hours for the MedleyDB

dataset, depending on the number of layers and the number of audio feature combinations.

The experimental results on the MUSDB18 dataset exceeded our expectations. The best accuracy with the Mel-spectrogram (93.60%) was about 20% higher than the models using ANNs. The results with the dB-MFCCs were also acceptable, and the CQT (using only its real part) demonstrated performance nearly equivalent to the Mel-spectrogram. In contrast, the CENS performed poorly, likely due to its image size, making it less suitable for musical instrument classification.

On the MedleyDB dataset, our models using CNNs consistently outperformed the models using ANNs. For pure-instrument classification, the accuracy (85.71%) was about 10% higher than with ANNs, and for all classes (54.22%) about 20% higher. Performance decreased when mixed-instrument classes were included. One possible reason is the artificial nature of our dataset derived from MedleyDB, as well as the use of Gaussian noise injection to increase the number of audio samples and balance the number of samples across classes.

Lastly, we examined experiments on the MedleyDB dataset using combined audio feature images after reshaping the features to have the same size, except for CENS. We did not conduct this experiment on the MUSDB18 dataset because it has already achieved an accuracy of 93.60%. So we focused more on the experiments on the MedleyDB dataset. These experiments led to improved performance, with the combination of the Mel-spectrogram and the dB-MFCCs giving the most balanced results. The final experiment, in which we replaced the Sigmoid with a Softmax activation function, yielded the best performance on the MedleyDB dataset. The combination of the Mel-spectrogram and the CQT also performed well. However, neither the three-feature combination nor the pair of the dB-MFCCs and the CQT outperformed these Mel-based two-feature combinations. This suggests that the Mel-spectrogram plays a primary role, while the dB-MFCCs and the CQT provide complementary information that improves performance on mixed-instrument classes.

# Chapter 6

# Conclusion

This chapter will provide a summary of our work, followed by a discussion of the limitations of our approach. Lastly, we will suggest feasible directions to improve musical instrument classification.

## 6.1 Summary of Our Work

In our work, we proposed methods to address the problem of multi-class, multi-label classification of musical instruments. In Chapter 1, we introduced Music Information Retrieval (MIR), musical instrument classification, and neural networks. Chapter 2 provided background information on Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs). It also detailed fundamental theories of the audio features we employed in this study. The Mel-spectrogram represents the audio energy distribution in a time-frequency domain, and the Mel-frequency cepstral coefficients (MFCCs) are a dimensionally reduced version of the Mel-spectrogram. The Constant-Q transform (CQT) represents frequencies in musical scale, and the Chroma Energy Normalized Statistics (CENS) is a digest version of CQT, which is efficient in describing musical phrases. The zero-crossing rate is effective in finding the noisiness of an audio signal.

We presented the preprocessing stage in Chapter 3. We first introduced the MUSDB18 and the MedleyDB datasets; the MUSDB18 dataset is relatively smaller and has fewer instruments than the MedleyDB dataset. We then described the audio feature extraction step. This chapter also presented Principal Component Analysis (PCA), Incremental PCA,

and upsampling techniques to conduct experiments efficiently.

Chapter 4 detailed the experiments on simple and hierarchical ANN models, and evaluation metrics are described. Chapter 5 detailed the experiments on models using CNNs with various combinations of audio feature images, and our proposed dB-MFCCs are introduced, which are rescaled on a logarithmic scale to enhance weak information in MFCCs. Experiments in each chapter are first conducted on the MUSDB18 dataset, and then further experiments are conducted on the MedleyDB.

Our study utilized multi-genre instrument mixtures derived from each dataset and investigated multi-class, multi-label classification models. The simple models using ANNs showed lower performance in classifying mixed-instrument classes, and the hierarchical models that were constructed using multiple simple models showed better performance than simple models. However, because of a trade-off between the depth of the hierarchy and the overall accuracy due to error propagation, their improvements were modest.

The models using CNNs outperformed the models using ANNs. But in the experiments on the MedleyDB dataset, the performance for classifying the mixed-instruments classes was unsatisfactory. This could be due to the artificial characteristics of our derived dataset. Employing combined audio feature images as input data to the models using CNNs improved the performance on the mixed-instruments classes. However, further improvements are needed. We expect that our approach would show better performance in the real-world situation.

## 6.2 Limitations of Our Approach

Our study has several limitations that constrain the immediate applicability of the proposed methods to real-world music.

1. Difficulty in obtaining proper datasets for musical instrument classification. As we focus on multi-class, multi-label classification, our requirements for the dataset were clear. (1) It must contain both mixture and isolated instrument tracks; (2) Its metadata

must provide detailed information for each piece of music; (3) It must cover various genres of music and a diverse set of instruments; and (4) The total number of pieces of music must be large. But it was difficult to find datasets that met all of our requirements. If a dataset contained large pieces of music, it contained only mixtures and no isolated tracks. If a dataset has large pieces of music with both mixture and isolated tracks, its metadata was ambiguous. We believe that we employed the best available datasets that meet our requirements. But the size of them are not satisfactory, which might lead to lower performance in some of the classification accuracy of our models.

2. Challenges in generating mixed audio clips close to real-world music. Music that we listen to is mixed and mastered by professional audio engineers. They always try to achieve a good balance among vocals and instruments so that musical messages are effectively delivered to listeners. But our procedure for generating mixed-instrument classes is very far from their methods, leading to imbalanced mixtures. Although it is possible to implement an automatic mixing of music, the balance means not only the balance of volume levels among instruments, but also the balance across all parts of music, such as intro, verse, chorus, and outro. This limitation was also a drawback of our experiments.

3. Lack of time to explore more audio features. We chose the audio features we employed in this study based on the performance from related works and our own musical experiences. But we should have explored some other audio features, which might have yield better performance or revealed new insights for MIR research. Though we proposed the dB-MFCCs, there may be other audio features that also fit the musical instrument classification.

## 6.3 Future Work

To further expand our work and address the limitations of our approach, we recommend several promising directions for future research:

1. Constructing a robust and adequate dataset: The most urgent need is to construct a high-quality dataset with a wide variety of instruments and a huge number of audio samples specifically designed for multi-class multi-label musical instrument classification across various genres. It would be much better if this dataset could be used as a benchmark standard so that researchers can easily compare their results with each other, because the currently available datasets are not appropriate for this purpose. The labelling must be more specific. For example, synth sounds must be categorized into synth bass, synth piano, synth string, and so on. It would also be better if each instrument's play style could be labelled in detail separately.

2. Developing an algorithm to generate good mixtures: Although developing an algorithm to generate mixtures that close to real-world music could be challenging, it might be easier than constructing a good dataset. Trying to emulate historically popular records might also be a good strategy, and generating various well-balanced mixtures could also be another form of upsampling.

3. Developing new features: Popular audio features, such as Mel-spectrogram and MFCCs, provide lots of information for MIR research. But they may miss some critical aspects of sound and music. Although Tzanetakis et al. [60] introduce the stereo panning features, there is still various information we have not utilized.

   Our proposed dB-MFCCs are based on our own experience of audio processing for music production. But we still need a much deeper understanding of how we listen to music and how it is produced in order to apply the knowledge to MIR research [14].

4. 3D clustering with space simulation: We can simulate a listening space through computer programming. As Gibson [18] mentioned, the space can be described using fre-

quencies, panning, and volume level, which correspond to height, width, and depth, and the target playback devices for produced music are usually stereo speakers; sound from each speaker is crossfed to both ears, which provides spatial perception [27]. Based on that information, music can be displayed in 3D space, where each instrument occupies its own place in this space. We can then apply clustering algorithms to classify musical instruments. The outputs from the clustering may contain additional information, such as delay or reverberation effects, which need postprocessing to finalize the classification, or those effect aspects can also be utilized for other purposes in MIR research.

These suggestions would improve musical instrument classification and could also be applied to a broader range of research related not only to music but also to any kind of audio and sound in general.

# Bibliography


[1] Charu C. Aggarwal. *Neural Networks and Deep Learning: A textbook*. Springer International Publishing, Cham, August 2018.

[2] P. Amutha and R. Priya. Evaluating the effectiveness of categorical encoding methods on higher secondary student's data for multi-class classification. *Tuijin Jishu/Journal of Propulsion Technology*, 44(6):6267–6273, December 2023.

[3] Jayme Garcia Arnal Barbedo and George Tzanetakis. Musical instrument classification using individual partials. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(1):111–122, March 2011.

[4] Daulappa G. Bhalke, C. B. Rama Rao, and Dattatraya S. Bormane. Automatic musical instrument classification using fractional fourier transform based- mfcc features and counter propagation neural network. *J. Intell. Inf. Syst.*, 46(3):425–446, June 2016.

[5] Rachel M. Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan P. Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 155–160, October 2014.

[6] Rachel M. Bittner, Julia Wilkins, Hanna Yip, and Juan P. Bello. Medleydb 2.0: New data and a system for sustainable data collection. In *Late-Breaking Demo Session of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, New York, NY, USA, August 2016.

[7] Maciej Blaszke, Gražina Korvel, and Bożena Kostek. Exploring neural networks for musical instrument identification in polyphonic audio. *IEEE Intelligent Systems*, 39(5):25–36, September 2024.

[8] Judith C. Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, January 1991.

[9] Michael A. Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, April 2008.

[10] Keunwoo Choi, György Fazekas, Kyunghyun Cho, and Mark Sandler. A tutorial on deep learning for music information retrieval. *arXiv:1709.04396*, May 2018.

[11] Keunwoo Choi, György Fazekas, Kyunghyun Cho, and Mark Sandler. A comparison of audio signal preprocessing methods for deep neural networks on music tagging. *arXiv:1709.01922*, February 2021.

[12] François Chollet. *Deep learning with python*. Manning Publications, New York, NY, April 2022.

[13] Madhuri Abhijit Darekar, Prasadu Peddi, and Sunayana Kundan Shivthare. Deep learning in musical instrument classification: Revolutionizing music information retrieval. *International Journal of Renewable Energy Exchange*, 12(4):49–54, January 2024.

[14] John Stephen Downie. Music information retrieval. *Annual Review of Information Science and Technology*, 37(1):295–340, January 2005.

[15] D.S.Shete and Prof. S. B. Patil. Zero crossing rate and energy of the speech signal of devanagari script. *IOSR journal of VLSI and Signal Processing*, 4:01–05, January 2014.

[16] Slim Essid, Gaël Richard, and Bertrand David. Musical instrument recognition by pairwise classification strategies. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1401–1412, July 2006.

[17] Soledad Galli and Christoph Molnar. *Python Feature Engineering Cookbook: A complete guide to crafting powerful features for your machine learning models*. Packt Publishing, August 2024.

[18] David Gibson. *The Art of Mixing: A Visual Guide to Recording, Engineering, and Production*. Taylor & Francis, January 2019.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 2016. `http://www.deeplearningbook.org`.

[20] Fabien Gouyon, François Pachet, and Olivier Delerue. On the use of zero-crossing rate for an application of classification of percussive sounds. In *Proceedings of the 3rd International Conference on Digital Audio Effects (DAFx-2000)*, December 2000.

[21] Siddharth Gururani, Mohit Sharma, and Alexander Lerch. An attention mechanism for musical instrument recognition. In *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands*, pages 83–90, November 2019.

[22] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, June 2011.

[23] Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1):208–221, January 2017.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016.

[25] Perfecto Herrera-Boyer, Xavier Amatriain, Eloi Batlle, and Xavier Serra. Towards instrument segmentation for music content description: a critical review of instrument classification techniques. In *Proceedings of the 1st International Symposium on Music Information Retrieval*. ISMIR, October 2000.

[26] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[27] Roey Izhaki. *Mixing Audio: Concepts, Practices, and Tools*. Taylor & Francis, London, UK, July 2023.

[28] Wenxin Jiang, Zbigniew W. Raś, and Alicja A. Wieczorkowska. *Clustering Driven Cascade Classifiers for Multi-indexing of Polyphonic Music by Instruments*, pages 19–38. Springer Berlin Heidelberg, Berlin, Heidelberg, March 2010.

[29] Ian T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, October 2002.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*, May 2015.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[32] Frank Kurth and Meinard Muller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):382–395, February 2008.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[34] Tao Li, Mitsunori Ogihara, and George Tzanetakis, editors. *Music Data Mining*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, Boca Raton, FL, May 2011.

[35] Vittorio Lippi and Giacomo Ceccarelli. Incremental principal component analysis: Exact implementation and continuity corrections. In *Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics*, page 473–480. SCITEPRESS - Science and Technology Publications, May 2019.

[36] Saranga Kingkor Mahanta, Nihar Jyoti Basisth, Eisha Halder, Abdullah Faiz Ur Rahman Khilji, and Partha Pakray. Exploiting cepstral coefficients and cnn for efficient musical instrument classification. *Evolving Systems*, 15(3):1043–1055, September 2023.

[37] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. *SciPy 2015*, July 2015.

[38] Jacob Murel. What is upsampling?, 2026. https://www.ibm.com/think/topics/upsampling, [Online; accessed 24-Feb-2026].

[39] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, March 2022.

[40] Meinard Müller, Frank Kurth, and Michael Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Conference on Music Information Retrieval*, pages 288–295. ISMIR, September 2005.

[41] Native Instruments GmbH. Stems. https://www.native-instruments.com/en/specials/stems/, [Online; accessed 21-Feb-2026].

[42] Mehmet Erdal Ozbek, Claude Delpha, and Pierre Duhamel. Musical note and instrument classification with likelihood-frequency-time analysis and support vector machines. In *Proceedings of the 15th European Signal Processing Conference (EUSIPCO 2007)*, pages 941–945, Poznan, Poland, September 2007.

[43] Taejin Park and Taejin Lee. Musical instrument sound classification with deep convolutional neural network using feature fusion approach. *arXiv:1512.07370*, December 2015.

[44] David M. W. Powers. What the F-measure doesn't measure: Features, flaws, fallacies and fixes. *arXiv:1503.06410*, September 2015.

[45] Lawrence Rabiner and Ronald Schafer. *Theory and Applications of Digital Speech Processing*. Pearson, 1st edition, November 2011.

[46] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, December 2017.

[47] Zbigniew W. Raś and Alicja A. Wieczorkowska. *Advances in Music Information Retrieval*. Studies in Computational Intelligence. Springer Berlin Heidelberg, March 2010.

[48] Lekshmi Chandrika Reghunath and Rajeev Rajan. Compact convolutional transformers for multiple predominant instrument recognition in polyphonic music. In *2024 9th International Conference on Communication and Electronics Systems (ICCES)*, pages 01–06, December 2024.

[49] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, November 1958.

[50] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[51] Karam Kumar Sahoo, Ridam Hazra, Muhammad Fazal Ijaz, Seongki Kim, Pawan Kumar Singh, and Mufti Mahmud. Mic_fuzzynet: Fuzzy integral based ensemble for automatic classification of musical instruments from audio signals. *IEEE Access*, 10:100797–100811, September 2022.

[52] Andrew H. Schwartz and Barbara G. Shinn-Cunningham. Effects of dynamic range compression on spatial selective auditory attention in normal-hearing listeners. *The Journal of the Acoustical Society of America*, 133(4):2329–2339, April 2013.

[53] Christian Schörkhuber and Anssi Klapuri. Constant-q transform toolbox for music processing. *Proc. 7th Sound and Music Computing Conf.*, January 2010.

[54] Dylan Sechet, Francesca Bugiotti, Matthieu Kowalski, Edouard d'Hérouville, and Filip Langiewicz. A hierarchical deep learning approach for minority instrument detection. In *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx-24)*, Guildford, UK, September 2024.

[55] Jonathon Shlens. A tutorial on principal component analysis. *arXiv:1404.1100*, April 2014.

[56] Christian Simmermacher, Da Deng, and Stephen Cranefield. Feature analysis and classification of classical musical instruments: an empirical study. In *Proceedings of the 6th Industrial Conference on Data Mining Conference on Advances in Data Mining: Applications in Medicine, Web Mining, Marketing, Image and Signal Mining*, ICDM'06, page 444–458, Berlin, Heidelberg, July 2006.

[57] Nicolas Sturmel, Antoine Liutkus, Jonathan Pinel, Laurent Girin, Sylvain Marchand, and Gaël Richard. Linear mixing models for active listening of music productions in realistic studio conditions. In *132nd Audio Engineering Society Convention*, Budapest, Hungary, April 2012.

[58] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, pages 10096–10106, July 2021.

[59] Richa Tiwari, Nirupama Shankar Babu T, Keshav Marda, Abhishek Mishra, Saahil Bhattar, and Anshika Ahluwalia. The impact of artificial intelligence in the workplace and its effect on the digital wellbeing of employees. *International Journal Of Progressive Research In Engineering Management And Science (IJPREMS)*, 4(6):2422–2427, June 2024.

[60] George Tzanetakis, Luis Gustavo Martins, Kirk McNally, and Randy Jones. Stereo panning information for music information retrieval tasks. *Journal of The Audio Engineering Society*, 58:409–417, June 2010.

[61] Fred Welsh. *Welsh's Synthesizer Cookbook: Synthesizer Programming, Sound Analysis, and Universal Patch Book*. Fred Welsh (author), January 2006.

[62] Guoqiang Wu and Jun Zhu. Multi-label classification: do hamming loss and subset accuracy really conflict with each other? In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20. Curran Associates Inc., December 2020.

[63] Jiaxiang Zheng, Moxi Cao, and Chongbin Zhang. Ickan: A deep musical instrument classification model incorporating kolmogorov-arnold network. *Scientific Reports*, 15(1):21573, July 2025.