VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF TELECOMUNICATION

DIGITAL SIGNAL PROCESSING COURSE PROJECT

TOPIC: Ensemble of Convolutional Neural Networks for Audio Classification Task

LECTURER: NGUYEN THANH TUAN

Num	Name	ID
1	Lam Thanh Phat	2111974
2	Le Khanh Huy	2110197

Ho Chi Minh City, August, 2023

TABLE OF CONTENTS

Contents

List of Tables

List of Figures

LIST OF ABBREVIATION

DSP: Digital Signal Processing
CNN: Convolutional Neuron Network
DFT: Discrete Fourier Transform
STFT: Short Time Fourier Transform

1 CHAPTER 1: INTRODUCTION

1.1 Introduction to Audio Classification Task

Audio classification is a fundamental problem in the field of machine learning and signal processing, involving the automatic categorization of audio data into predefined classes or categories based on their content. The primary goal of audio classification is to develop intelligent systems that can recognize and differentiate between various types of audio signals, such as speech, music, environmental sounds, and specific sound events. The ability to classify audio has become increasingly important due to the growing popularity of applications that heavily rely on audio data analysis. From speech recognition in virtual assistants and language translation systems to music genre identification on streaming platforms and acoustic event detection in surveillance, audio classification plays a pivotal role in enabling computers to interpret and respond to auditory information. Unlike visual data, which is represented by images with spatial structures, audio is represented as a time-varying waveform. This temporal nature makes it essential to capture relevant patterns and features that convey information about the underlying content. Additionally, audio data may exhibit variations in pitch, timbre, intensity, and background noise, making the task of classification even more intricate.

In recent years, significant advancements in machine learning, particularly with deep learning models, have revolutionized audio classification. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), etc, have proven to be highly effective in extracting discriminative features from audio signals and learning complex patterns that lead to accurate classification. Audio Classification holds great potential for real-world applications. From improving human-computer interaction and understanding the acoustic environment to enhancing multimedia content organization and enriching the user experience, it continues to be a vibrant and essential area of research and development.

1.2 Project Overview

In this project, we will create a deep learning model that uses neural networks to classify the genre of music the available labeled dataset and create a website platform to facilitate the audio classification task automatically. The project's workflow includes these steps below:

- From initial audio dataset, we perform digital audio signal processing and feature extraction to transform raw data into numerical features that can be processed by neuron networks while preserving the information in the original dataset. In this case, features extracted from raw data will be converted into images and fed to convolutional neuron network (CNN). Hence, audio classification task is now considered as Image classification task.
- Create multiple CNN classifiers to categorize processed images into classes defined by the initial dataset. Parallelly, we consider evaluation metrics of these CNN classifiers during the training phase to assess each model's performance. After that, we present an ensemble of these classifiers for the inference phase.
- Deploy model inference on the web to make it accessible and usable for users. By doing so, you enable users to interact with the model and obtain predictions or insights without having to install any specialized software or have knowledge of the underlying ML algorithms. In addition, the classification results from the audios uploaded by the user can also be saved for other further purposes.

2 CHAPTER 2: BACKGROUND

2.1 Audio Signal

An audio signal is a representation of sound. It encodes all the necessary information required to reproduce sound. Audio signals come in two basic types: analog and digital. In the real world, conversions between digital and analog waveforms are common and necessary. ADC (Analog-to-Digital Converter) and the DAC (Digital-to-Analog Converter) are part of audio signal processing and they achieve these conversions.

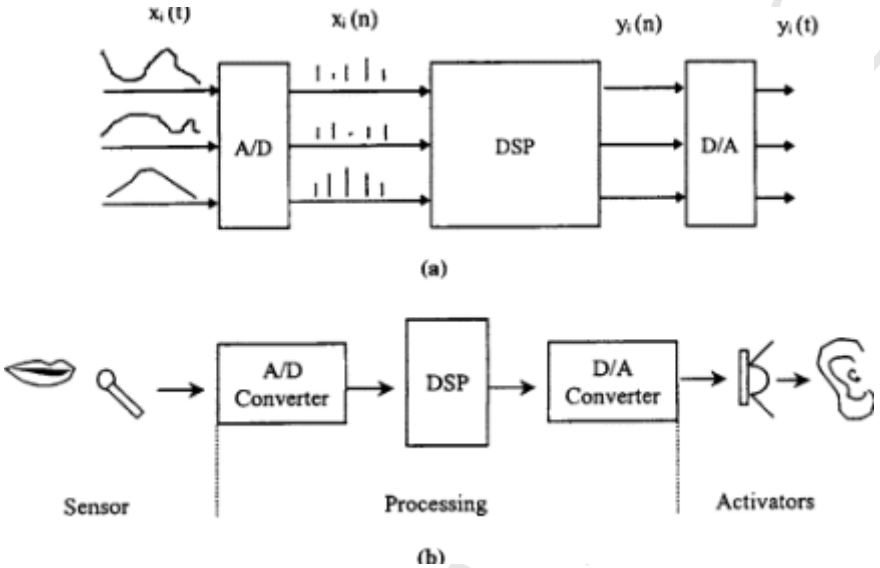

Figure 1: An DSP system for Audio signal.

2.1.1 Audio Feature Extraction

Audio feature extraction is a crucial process in the field of digital signal processing and machine learning, where relevant characteristics and patterns are extracted from raw audio signals to facilitate further analysis and understanding. These features serve as essential inputs for various audio-related applications, such as speech recognition, music classification, sound event detection, and more. The raw audio signal is essentially a continuous waveform, which is challenging to process directly due to its high dimensionality and complex nature. Audio feature extraction aims to reduce this complexity while retaining essential information.[1]

Audio features are descriptions of sound or an audio signal that can be fed into statistical or ML models to build intelligent audio systems. Audio applications that use such features include audio classification, speech recognition, etc.

At the level of abstraction aspect, these broad categories cover mainly musical signals rather than audio in general and can be separated into these types below:

- High-level: These are the abstract features that are understood and enjoyed by humans.These include instrumentation, key, chords, melody, harmony, rhythm, genre, mood, etc.
- Mid-level: These are features we can perceive. These include pitch, beat-related descriptors, note onsets, fluctuation patterns, MFCCs, etc. We may say that these are aggregations of low-level features.
- Low-level: These are statistical features that are extracted from the audio. These make sense to the machine, but not to humans. Examples include amplitude envelope, energy, spectral centroid, spectral flux, zero-crossing rate, etc.

At the signal domain aspect, audio features can be categorized into the following types:

- Time domain: These are extracted from waveforms of the raw audio. Zero crossing rate, amplitude envelope, and RMS energy are examples.
- Frequency domain: These focus on the frequency components of the audio signal. Signals are generally converted from the time domain to the frequency domain using the Fourier Transform. Band energy ratio, spectral centroid, and spectral flux are examples.
- Time-frequency representation: These features combine both the time and frequency components of the audio signal. The time-frequency representation is obtained by applying the Short-Time Fourier Transform (STFT) on the time domain waveform. Spectrogram, mel-spectrogram, and constant-Q transform are examples.

All features carry information about many aspects of the data samples. Depending on different types of specific problems, different appropriate features will be extracted where relevant characteristics and patterns are extracted from raw audio signals to facilitate further analysis and understanding as well as to enhance the model's performance. In this project, we utilize audio features at the signal domain aspect, especially Time-frequency representation. Spectrogram and Mel-spectrogram are extracted and converted into image format for neuron network classifiers.

2.1.2 Discrete Fourier Transform

The discrete Fourier transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency.

The discrete Fourier transform transforms a sequence of N complex numbers $\{x_n\} = x_0, x_1, ...x_{N-1}$ is defined as $\{X_n\} = X_0, X_1, ..., X_{N-1}$ where: $\{X_k\} = \sum_{n=0}^{N-1} x_n . e^{-\frac{2\pi}{N} kn}$

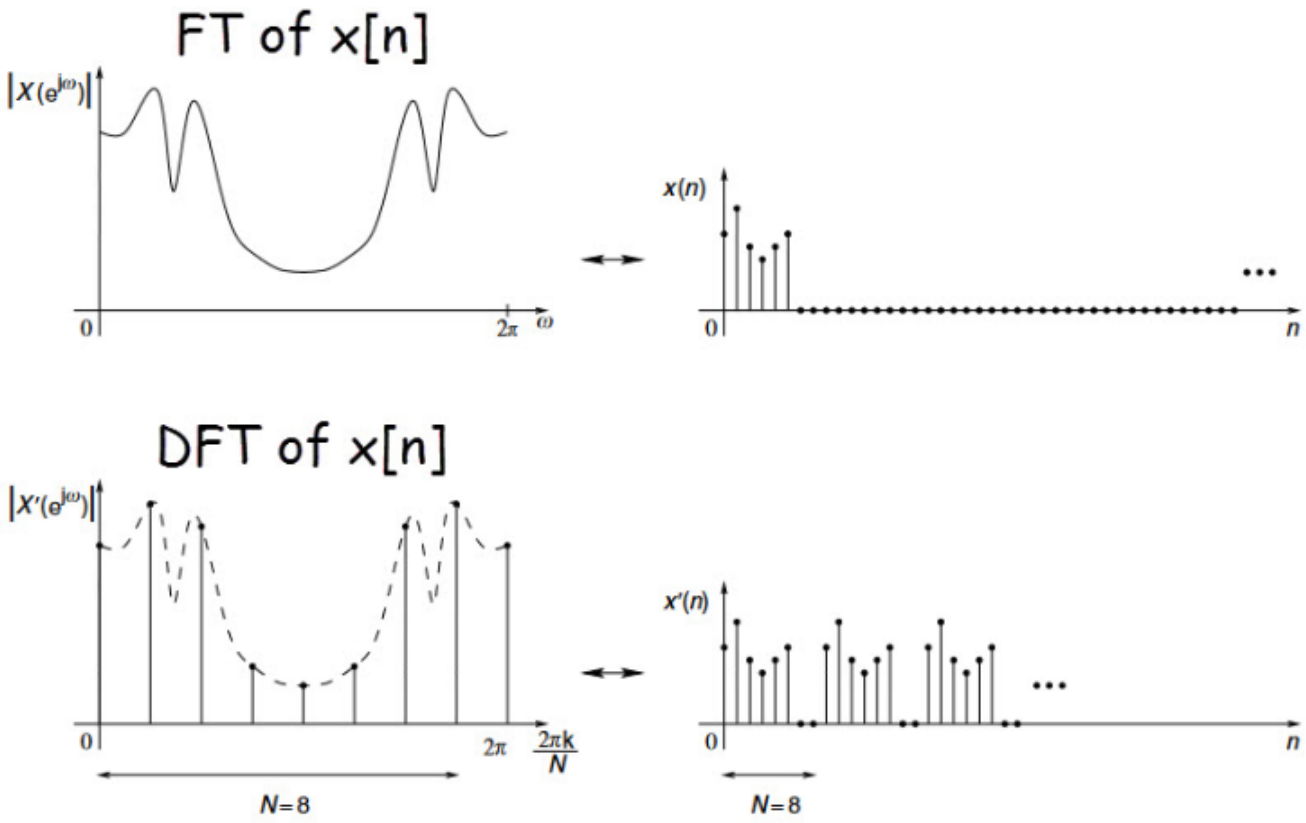

Figure 2: Figure 2: DFT visualization

DFT provides a full-frequency representation of the entire signal, analyzing the signal's frequency content over its entire duration. The DFT is typically computed over the entire signal, and the resulting frequency spectrum is discrete, providing information about the signal's frequency components at equally spaced frequency bins.

2.1.3 Short Time Fourier Transform

Notice that DFT shows the presence of different frequency components in the original audio signal sample into a magnitude spectrum plot, but it still provides global spectral information as a picture that indicates the average presence of frequency components across the whole duration of the audio signal. As a result, we do not know when a certain frequency component presents more or less at a certain period of time. The STFT is an extension of the DFT that analyzes the frequency content of a signal over short overlapping time windows. It allows for a time-varying analysis of the signal's frequency content, capturing changes in the frequency spectrum as the signal evolves over time.

The STFT breaks down the signal into small segments or frames and computes the DFT separately for each frame. By using overlapping frames, it provides a trade-off between temporal and spectral resolution, offering insights into the signal's frequency content at different time points. The resulting representation is a time-frequency spectrogram, which shows how the frequency components of the signal change over time.

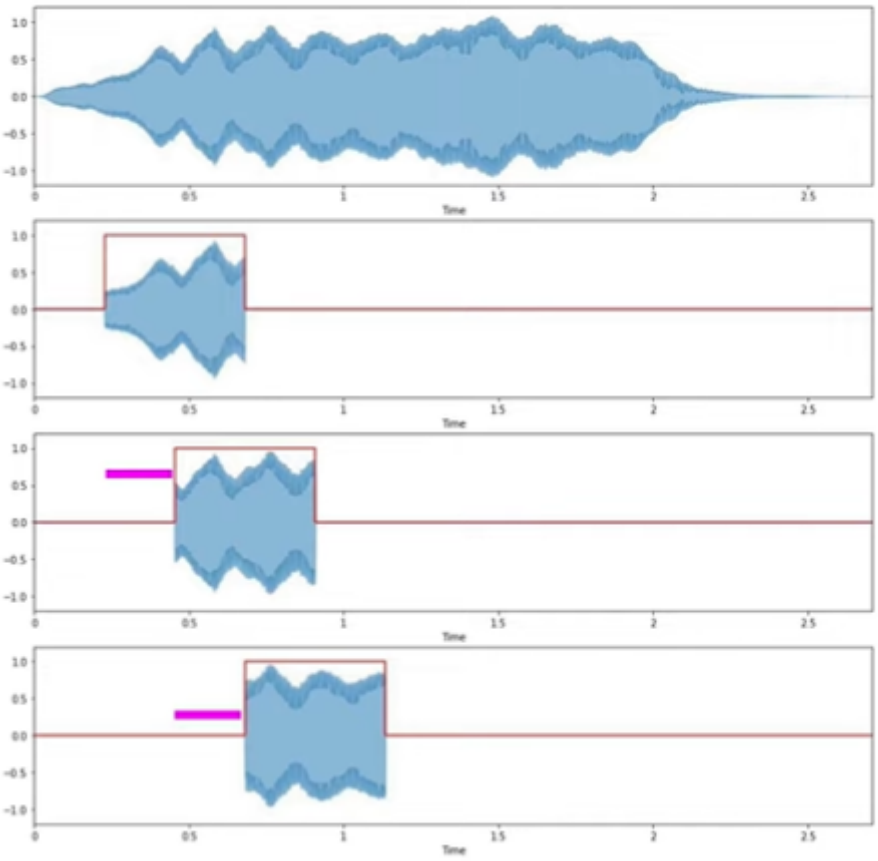

Figure 3: STFT window

Mathematically, STFT expression can be obtained as below equation:

$$STFT(m,k) = \sum_{n=0}^{N-1} x(n+mH).w(n).e^{-i\frac{2\pi}{N}kn}$$

In the expression above, k frequency bins of a single DFT will be calculated in the current frame number n, N is the number of samples at each DFT calculating, H is the hop size which refers to the displacement or shift between consecutive frames when computing the Fourier Transform for each segment of the signal, mH is the current sample of current frame, $w(n)$ is the value of Window function. A window often used in signal processing is the Hann window which equation is $w(n) = 0.5\left(1 - cos\left(2\pi\frac{n}{N}\right)\right)$.

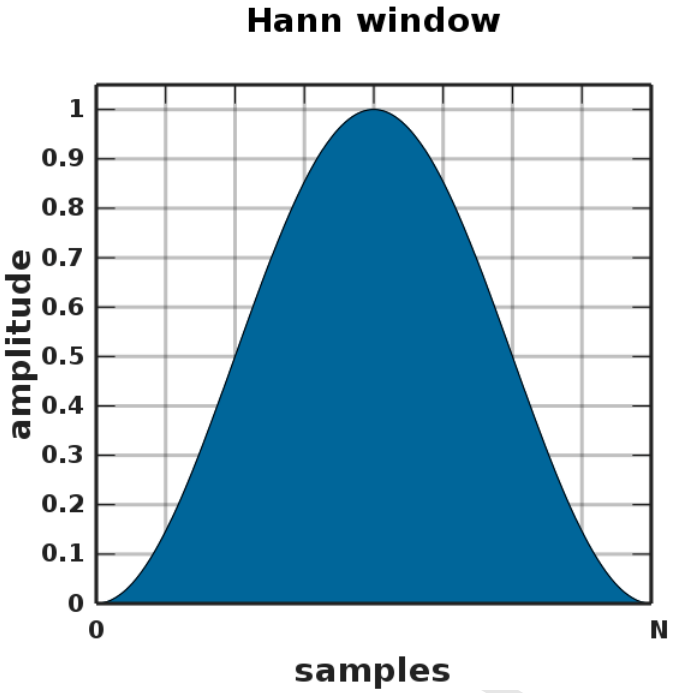

Figure 4: Hann window

STFT's result is a spectral matrix of complex Fourier coefficients which column describes the DFT of the audio signal at a single time frame. So its shape is (*NB,NF*) = (number of frequency bins, number of frames).

Number of frequency bins from taking DFT at each time frame is N, but as a property of DFT, from the Nyquist frequency bin $\frac{N}{2}$, each Fourier complex coefficient at frequency bin $j\left(j > \frac{N}{2}\right)$ conjugates with the Fourier complex coefficient at frequency bin $i\left(i < \frac{N}{2}\right)$ given the constraint $i + j = N$, $i \neq 0$, while the magnitudes of these two coefficients are similar. Because we only focus on the magnitude of DFT, we ignore frequency bins greater than Nyquist frequency bin $\frac{N}{2}$. Hence, the number of frequency bins is reduced to NB = $\frac{N}{2} + 1$.

Number of frames is calculated by NF = $Floor\left(\frac{L-W}{H}\right) + 1$, where L is the length of sampling audio signal, W is window size, H is hope size.

2.1.4 Spectrogram

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. It is a commonly used tool in the fields of digital signal processing, audio analysis, and speech processing.[2]

To create a spectrogram, the Short-Time Fourier Transform (STFT) is typically applied to a time-domain signal. The magnitude of the complex numbers in the spectral matrix obtained from the STFT is computed, as it represents the strength of each frequency component of the signal in the spectrogram, brighter colors or higher intensities represent stronger frequency components, while darker colors or lower intensities indicate weaker or absent frequency components.

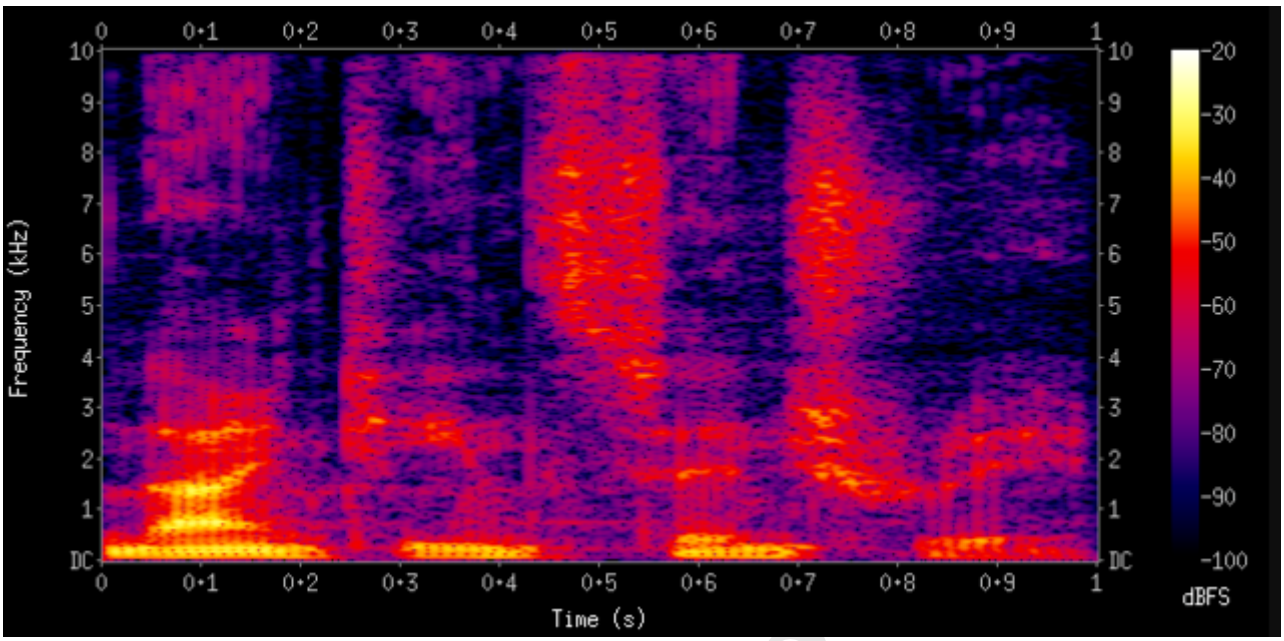

Figure 5: A spectrogram of audio signal

2.1.5 Mel-spectrogram

A Mel-spectrogram is a variant of the traditional spectrogram that utilizes the Mel scale to map frequencies into a more perceptually meaningful representation. The Mel scale is a perceptual scale of pitches, designed to mimic the non-linear human auditory system's response to different frequencies. Unlike the linear scale used in the traditional spectrogram, the Mel scale is logarithmically spaced, with more resolution at lower frequencies and less resolution at higher frequencies. This makes it more suitable for capturing the way humans perceive sound.

A formula (O'Shaughnessy 1987) to convert (f) hertz into (m) meals is given as:

$$m = 2595 log_{10}\left(1 + \frac{f}{700}\right) = 1127 \ln\left(1 + \frac{f}{700}\right)$$

To obtain Mel-spectrogram, we follow these steps:

- The Short-Time Fourier Transform (STFT) is typically applied to a time-domain signal. Instead of directly computing the magnitude spectrum.
- Create triangular filter bank. The Mel spectrogram uses a set of triangular filters (known as the Mel filterbank) that are spaced along the Mel scale to obtain the frequency responses. Each triangular filter corresponds to a specific Mel frequency bin.
- The filterbank multiplies the magnitudes from the STFT with the filter's response for each frequency bin. The result is the energy distribution across different Mel frequency bins.
- The energy values obtained from the filterbank are usually transformed using a logarithm to compress the dynamic range. This is typically done using the natural logarithm (logarithm base e) or the base-10 logarithm STFT with the filter's response for each frequency bin. The result is the energy distribution across different Mel frequency bins.

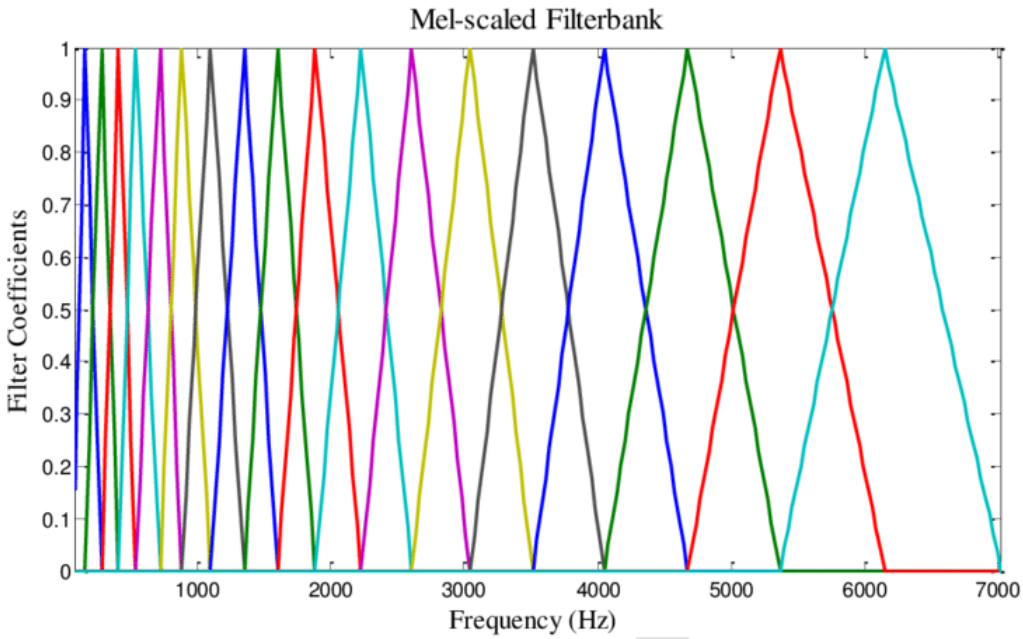

Figure 6: Triangular Mel filter bank

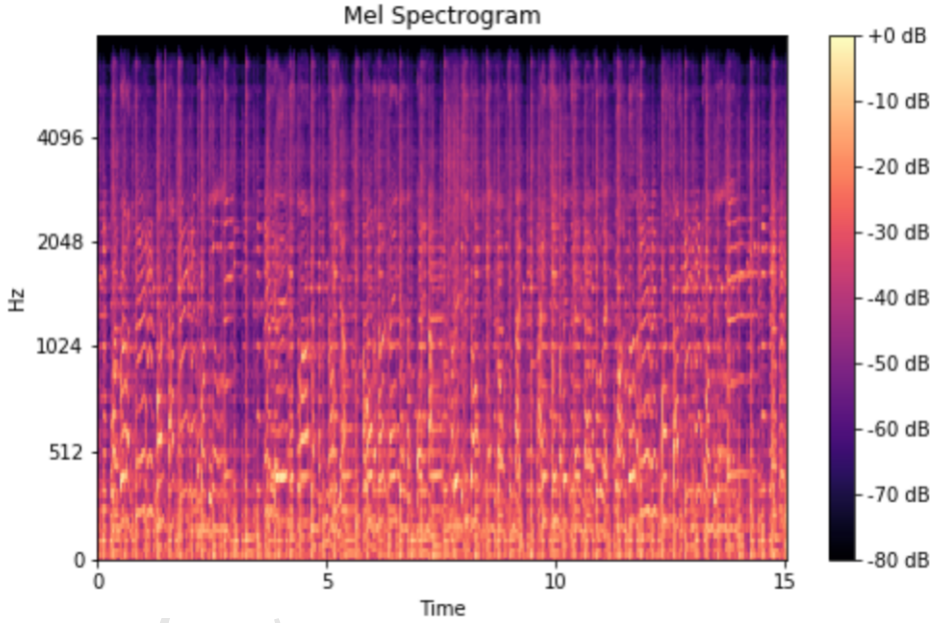

Figure 7: A Mel-spectrogram of audio signal

2.2 Convolutional Neural Network (CNN)

In deep learning, a convolutional neural network (CNN) is a class of artificial neural network most commonly applied to analyze visual imagery. CNNs use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers. They are specifically designed to process pixel data and are used in image recognition and processing.

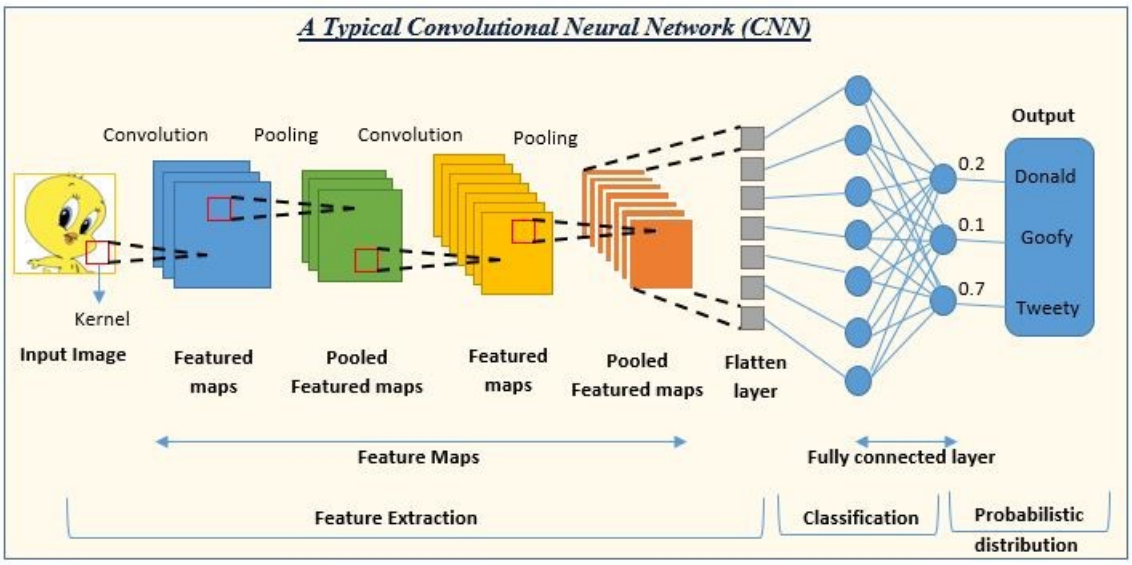

Figure 8: A typical CNN Artchitecture

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (holding the class scores) through a differentiable function. A few distinct types of layers are commonly used. These are further discussed below.

2.2.1 Convolutional Layers

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter entries and the input, producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

The spatial size of the output volume is a function of the input volume size W, the kernel size K of the convolutional layer neurons, the stride S, and the amount of zero padding P on the border. The number of neurons that "fit" in a given volume is then: [3]

$$\frac{W-K+2P}{S}+1$$

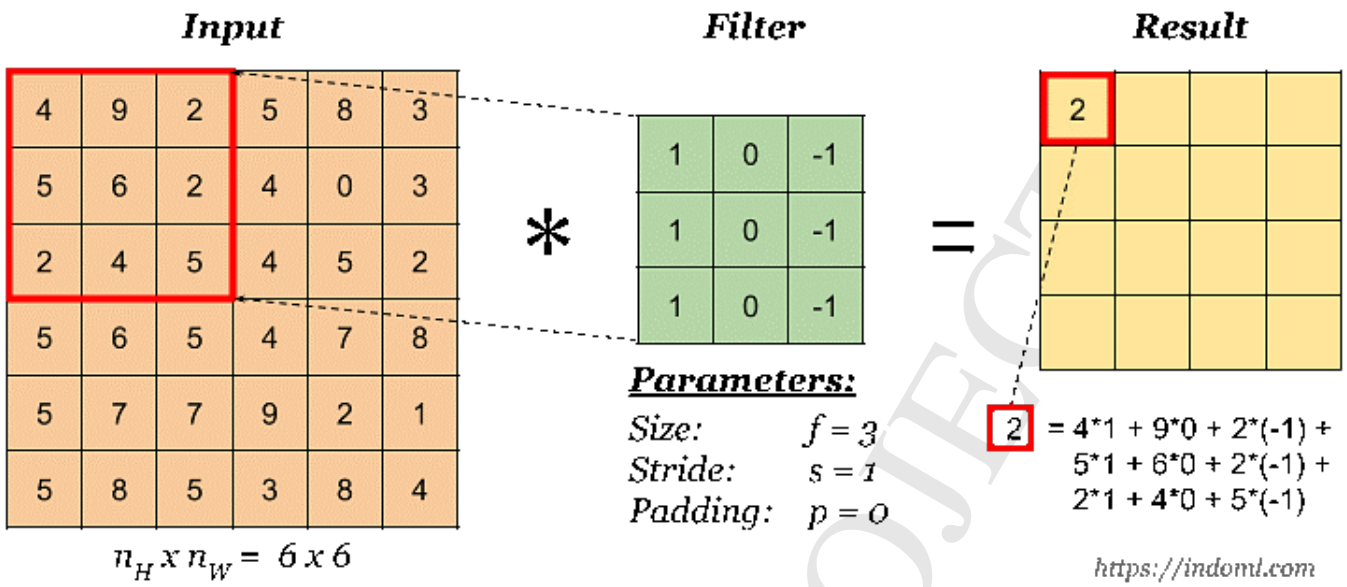

Figure 9: A Convolution Operation

2.2.2 Pooling Layers

Pooling layers are an essential component in Convolutional Neural Networks (CNNs) and other deep learning models used in computer vision tasks. They help reduce the spatial dimensions of the feature maps while retaining the most relevant information. Pooling layers play a crucial role in managing computational complexity, controlling overfitting, and introducing translational invariance in the network. The two most common types of pooling layers are Max Pooling, Average Pooling.

2.2.3 Activation Function

An activation function is a critical component of artificial neural networks, including deep learning models like Convolutional Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs). It introduces non-linearity to the network, enabling it to learn complex patterns and relationships within the data. Without activation functions, the neural network would behave like a linear model, making it limited in its ability to approximate non-linear functions and solve complex tasks. Some activation functions are shown in below figure:

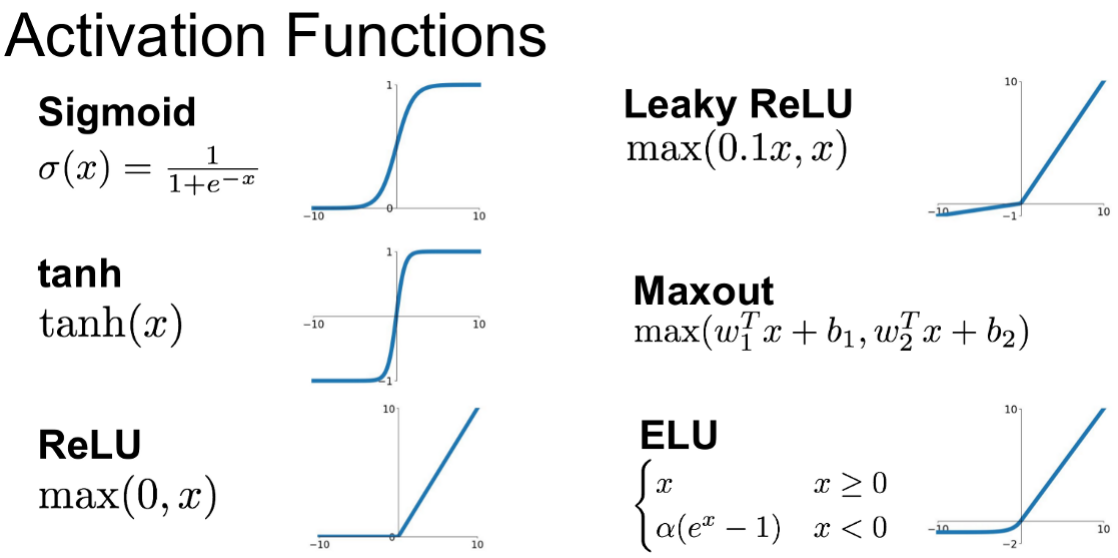

Figure 10: Activation function

2.2.4 Fully-connected Layers

A Fully Connected Layer, also known as a Dense Layer, is a type of layer commonly used in neural networks, including Convolutional Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs). It is the simplest and most standard layer type where each neuron in the layer is connected to every neuron in the previous layer. In a Fully Connected Layer, the input to each neuron is the output from all the neurons in the previous layer, and each neuron has its own set of learnable weights and biases. The output of the layer is obtained by performing a weighted sum of the inputs, followed by the application of an activation function.

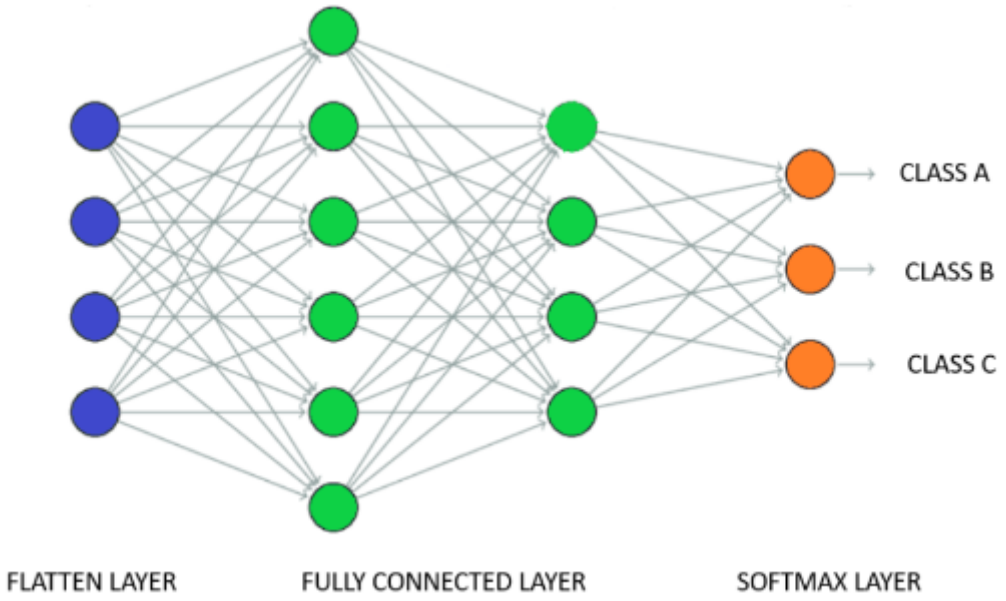

Figure 11: Fully-connected Layers in CNN

Fully Connected Layers are commonly used as the final layers in neural networks for tasks such as classification and regression. In classification tasks, the last Fully Connected Layer is often followed by a softmax activation function to produce a probability distribution over different classes. The class with the highest probability is then chosen as the predicted class.

2.2.5 Batch Normalization

Batch normalization (also known as batch norm) is a method used to make training of artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling. It was proposed by Sergey Ioffe and Christian Szegedy in 2015. BatchNorm is typically applied to the output of a layer, before the activation function, and is often used after convolutional layers or fully connected layers. It normalizes the activations of a batch of data by subtracting the batch mean and dividing by the batch standard deviation. The benefits of BatchNorm include Stability and Faster Convergence, Regularization, Less Sensitive to Weight Initialization and Generalization. The below figure shows steps for Batch Normalization: [4]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 12: Batch Normalization

2.2.6 Dropout

Dropout is a regularization technique used in deep learning to prevent overfitting and improve the generalization of a neural network. It involves randomly deactivating (dropping out) a fraction of neurons or units during each training iteration, effectively removing them from the network temporarily. Dropout was introduced as a regularization method by Geoffrey Hinton and his colleagues in 2012. The main idea behind dropout is to prevent the neurons from relying too much on each other during training. By randomly dropping out neurons, the network is forced to learn more robust and independent representations. This helps to reduce overfitting, as the network cannot memorize specific patterns or noise present in the training data.

2.2.7 Loss Function

In machine learning and deep learning, a loss function, also known as a cost function or objective function, is a measure of how well a model's predictions match the actual ground truth labels or targets. The primary goal during training is to minimize the value of the loss function, which represents the discrepancy between the predicted outputs and the true values.

The choice of the loss function depends on the specific task and the type of problem being addressed, such as classification, regression, or other specialized tasks. Different loss functions are used for different types of problems: Mean Squared Error (MSE) Loss , Binary Cross-Entropy (Log Loss) Loss , Categorical Cross-Entropy Loss, etc.

In this project, we use Categorical Cross-Entropy Loss to evaluate the model performance. It is also called Softmax Loss. It is a Softmax activation plus a Cross-Entropy loss. If we use this loss, we will train a CNN to output a probability over the C classes for each image. It is used for multi-class classification. Given **F** = $[f(s)_1, f(s)_2, ..., f(s)_c]$ is the output of Softmax Layer which input is S. **T** = $[t_1, t_2, ..., t_C]$ is the target one-hot vector. Categorical Cross-Entropy Loss formula is shown as below:[5]

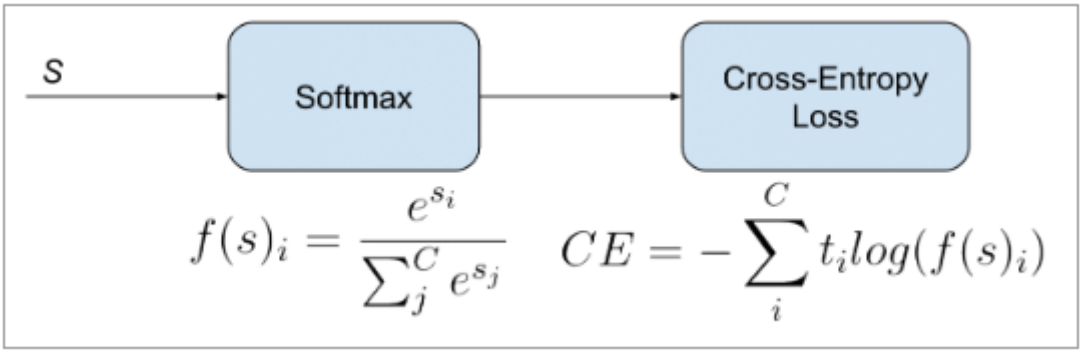

Figure 13: Categorical Cross-Entropy Loss

2.2.8 Optimizer

An optimizer is an algorithm or method used to update the parameters of a machine learning model during the training process. The goal of optimization is to minimize (or maximize, in certain cases) a specified objective function, also known as the loss function or the cost function, which measures the discrepancy between the model's predictions and the true values. In this project, we use Adam as an optimizer for all classifiers. Adaptive Moment Estimation (ADAM) is an algorithm for optimization techniques for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with the momentum algorithm and the 'RMSP' algorithm.

2.3 Muti-class classification metrics

Section 2.3 is mostly taken from [6]

2.3.1 Accuracy

Accuracy is a popular performance metric in classification problems. The good news is that you can directly borrow the metric from binary classification and calculate it for multi-class in the same way. Accuracy measures the proportion of correctly classified cases from the total number of objects in the dataset. To compute the metric, divide the number of correct predictions by the total number of predictions made by the model.

$$Accuracy = \frac{Correct\,prediction}{All\,prediction}$$

2.3.2 Precision

Precision for a given class in multi-class classification is the fraction of instances correctly classified as belonging to a specific class out of all instances the model predicted to belong to that class.

$$Precision_{ClassA} = \frac{TP_{ClassA}}{TP_{ClassA} + FP_{ClassA}}$$

2.3.3 Recall

Recall in multi-class classification is the fraction of instances in a class that the model correctly classified out of all instances in that class.

$$Recall_{ClassA} = \frac{TP_{ClassA}}{TP_{ClassA} + FN_{ClassA}}$$

2.4 Libraries and tools

2.4.1 Tensorflow

TensorFlow is an open-source deep learning framework developed and maintained by the Google Brain team. As one of the most popular and powerful machine learning libraries, TensorFlow has revolutionized the field of artificial intelligence, enabling researchers and developers to build sophisticated and scalable deep learning models for a wide range of applications. At its core, TensorFlow is designed to work with data represented as multi-dimensional arrays called tensors, which are capable of handling diverse data types and structures. Its flexible and efficient computational graph architecture allows users to construct complex mathematical operations and neural network architectures, making it ideal for tackling challenging machine learning tasks. In TensorFlow, CNN can be built using the high-level API tf.Keras, which provides an intuitive and user-friendly interface for designing and training neural networks. Here's a step-by-step guide to creating a simple CNN for image classification using TensorFlow and tf.Keras.

2.4.2 Librosa

Librosa is a Python library designed for music and audio analysis. It provides a wide range of tools and functions for extracting meaningful features from audio data, enabling researchers, musicians, and developers to perform various audio processing tasks. Librosa is widely used in fields such as music information retrieval, audio signal processing, speech processing, etc.

2.4.3 Streamlit

Streamlit is an open-source Python library that simplifies the process of creating web applications for data science and machine learning projects. Streamlit allows data scientists and developers to turn data scripts into interactive web applications with minimal effort. With its intuitive API and straightforward design, Streamlit has become a popular choice for quickly sharing and visualizing data insights with others.

The main objective of Streamlit is to streamline the process of building web applications, making it accessible to a broader audience, including those without extensive web development experience. By using Streamlit, users can focus on their data analysis and machine learning tasks, while leaving the web application creation process to the library. Its lightweight and user-friendly approach has made it a valuable tool for data scientists, machine learning engineers, and anyone looking to share data-driven insights through interactive web applications.

3 CHAPTER 3: THE PROPOSED WORK

3.1 Introduction

3.1.1 Overall pipeline

In this project, given the audio dataset collected from Intenet, we propose an end-to-end pipeline to perform audio classification task shown below. We divide the whole process into 3 stages: Data Processing, Model building and Evaluation, Deploying.

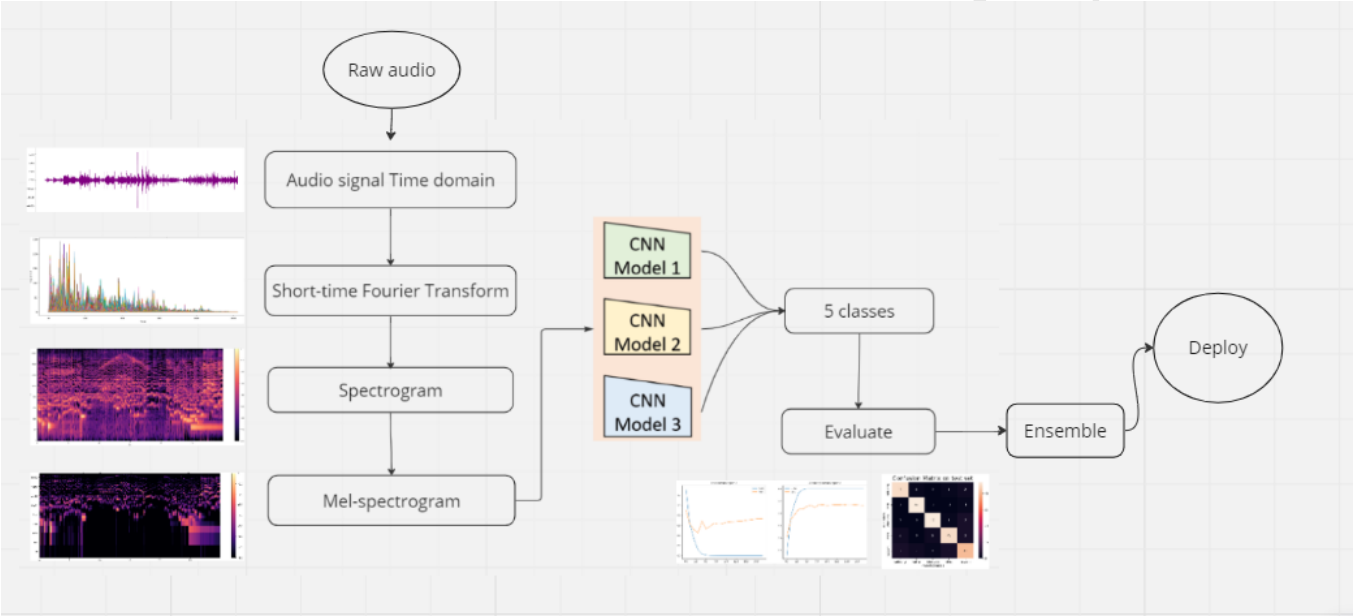

Figure 14: Sound classification process

3.1.2 About the dataset

In this project, we work on Vietnam Traditional Music (5 genres) dataset[7], which was collected from Kaggle. The dataset includes 2500 audio files of 5 classes: cailuong, catru, chauvan, cheo, hatxam. Each class includes 500 wav files with a length of about 30s. This dataset can be presented as following tree folder structure:

```
├── rawdata
│   ├── cailuong
│   │       ├── Cailuong000.wav
│   │       ├── Cailuong001.wav
│   │       ├── Cailuong002.wav
│   │       ├── ...
│   ├── catru
│   │       ├── Catru000.wav
│   │       ├── Catru001.wav
│   │       ├── Catru002.wav
│   │       ├── ...
│   ├── chauvan
│   │       ├── Chauvan000.wav
│   │       ├── Chauvan001.wav
│   │       ├── Chauvan002.wav
│   │       ├── ...
│   ├── cheo
│   │       ├── Cheo000.wav
│   │       ├── Cheo001.wav
│   │       ├── Cheo002.wav
│   │       ├── ...
│   ├── hatxam
│   │       ├── Hatxam000.wav
│   │       ├── Hatxam001.wav
│   │       ├── Hatxam002.wav
│   │       ├── ...
```

Figure 15: Dataset tree folder structure

3.2 Data Processing

Firstly, we extract labels from all audio files based on their respective folder name and filename. We define a type_index list to assign each class with an index for further usage.

```
# Define variables for further use
type_list = {0: ["cailuong", "CaiLuong"], 1: ["catru", "Catru"],
             2:["chauvan", "Chauvan"], 3: ["cheo", "Cheo"], 4: ["hatxam",
"Xam"]}
class_list = {0: "cailuong", 1: "catru", 2:"chauvan", 3: "cheo", 4: "hatxam"}
```

Figure 16: Extract label from all audio files

For each class, we define a Class called processing to store all necessary attributes and method functions for the end-to-end process of that class. For instance, Preprocessing can be defined as follow:

```
class Preprocessing():
    def __init__(self, root, save_root, dataset_root, train_root,
                 val_root, test_root, samples, type_index, num_of_samples):
        self.root = root                              # Directory of folder
        self.save_root = save_root                    # Directory of folder saving mel-spec images
        self.dataset_root = dataset_root              # Directory to save dataset
        self.train_root = train_root                  # Directory of trainset
        self.val_root = val_root                      # Directory of valset
        self.test_root = test_root                    # Directory of test_root
        self.samples = samples                         # Dictionary store information of each class folder
        self.type_index = type_index                  # Class index
        self.num_of_samples = num_of_samples          # Num of samples in class to process
```

Figure 17: Preprocessor definition

Note that, the attribute self.samples is a dictionary which key is the index of a sample in the class, and value is another dictionary storing information of that sample such as directory, sampling array, stft, spectrogram coefficients, mel-spectrogram coefficients. For example:

```
self.samples{356:{'dir':'/content/drive/MyDrive/DATA/VNTM3/chauvan/Chauvan.356
.wav', 'sampling': array([0.0682373 , 0.06588745, 0.04116821, ..., 0.03323364,
0.03179932,        0.02005005], dtype=float32) …}
```

Figure 18: Self.samples

3.2.1 Sampling

From raw audio file with .wav format, we perform signal sampling using librosa.load function with sampling rate sr = 22050 which result in a sampling array. The size of array is the length of the sampling signal.

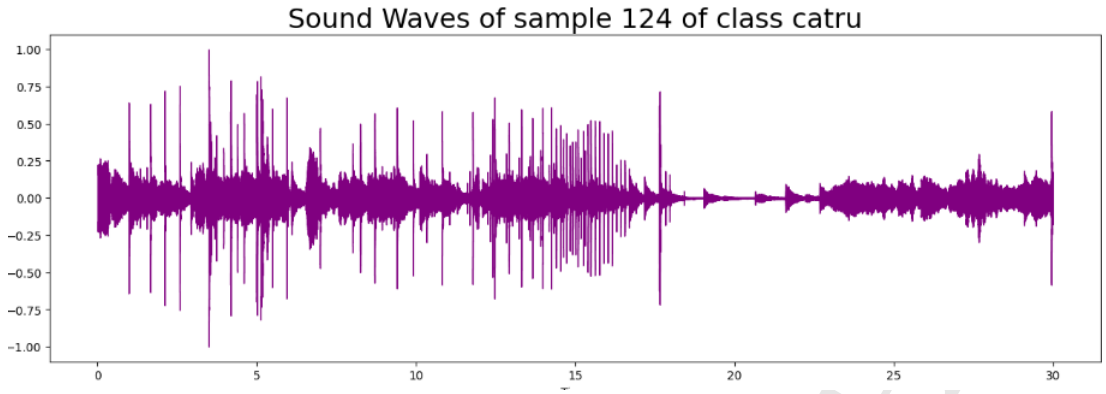

Figure 19: Time domain visualization of a sample

3.2.2 Short Time Fourier Transform

Get STFT of audio signal using librosa.stft. We define n_fft, hop_length, window equal to 2048, 512, "Hann" respectively. The function returns a spectral matrix, which shape is (NB, NF) as explained in section 2.1.

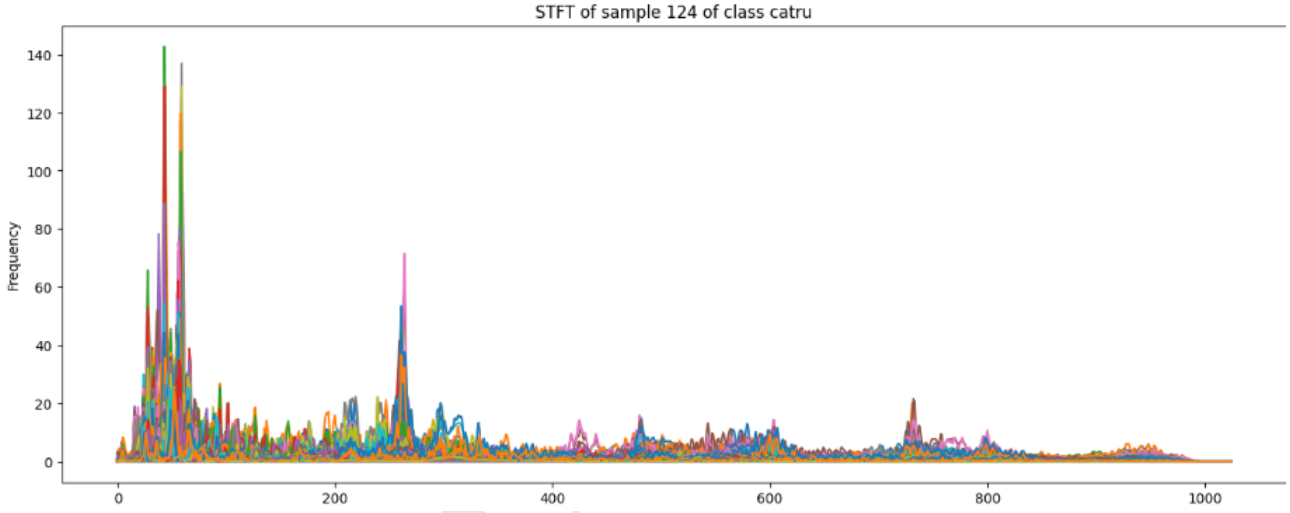

Figure 20: Frequency spectrum of a sample

3.2.3 Spectrogram

Extracted spectrogram spectral matrix by using librosa.feature.melspectrogram function, we defined value of arguments similar to the above STFT settings.

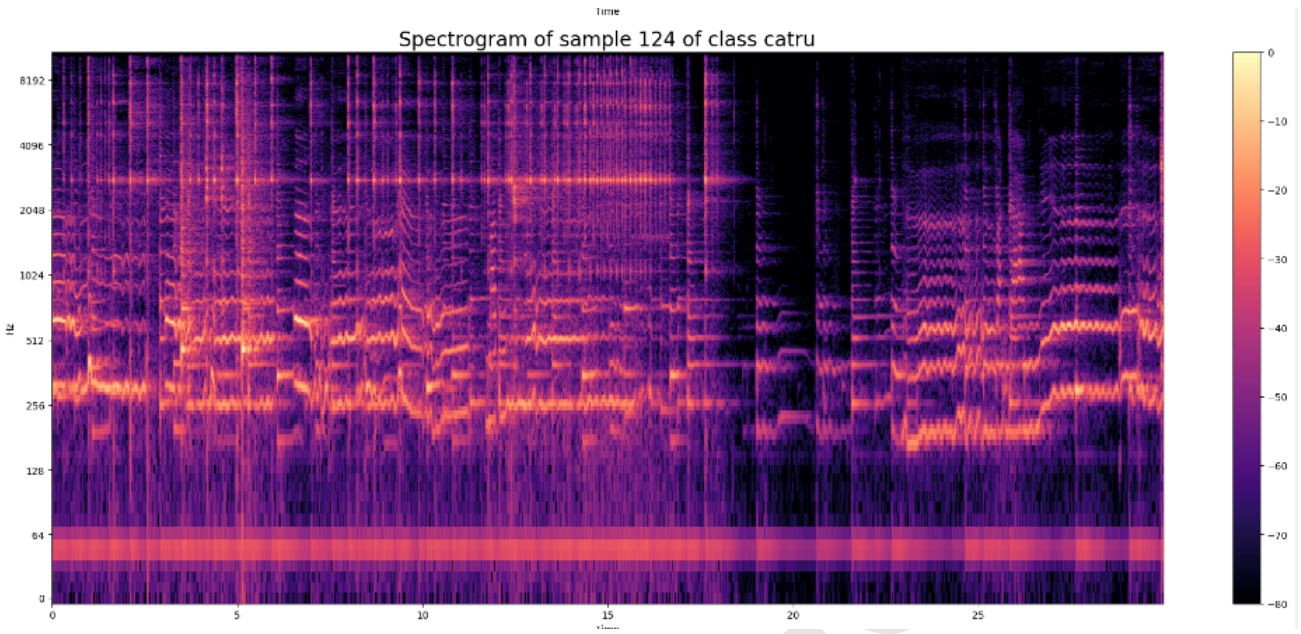

Figure 21: Spectrogram plot of a sample

3.2.4 Mel-spectrogram

Similarly, we extract mel-spectrograms by using lb.feature.melspectrogram function with the same argument value settings.

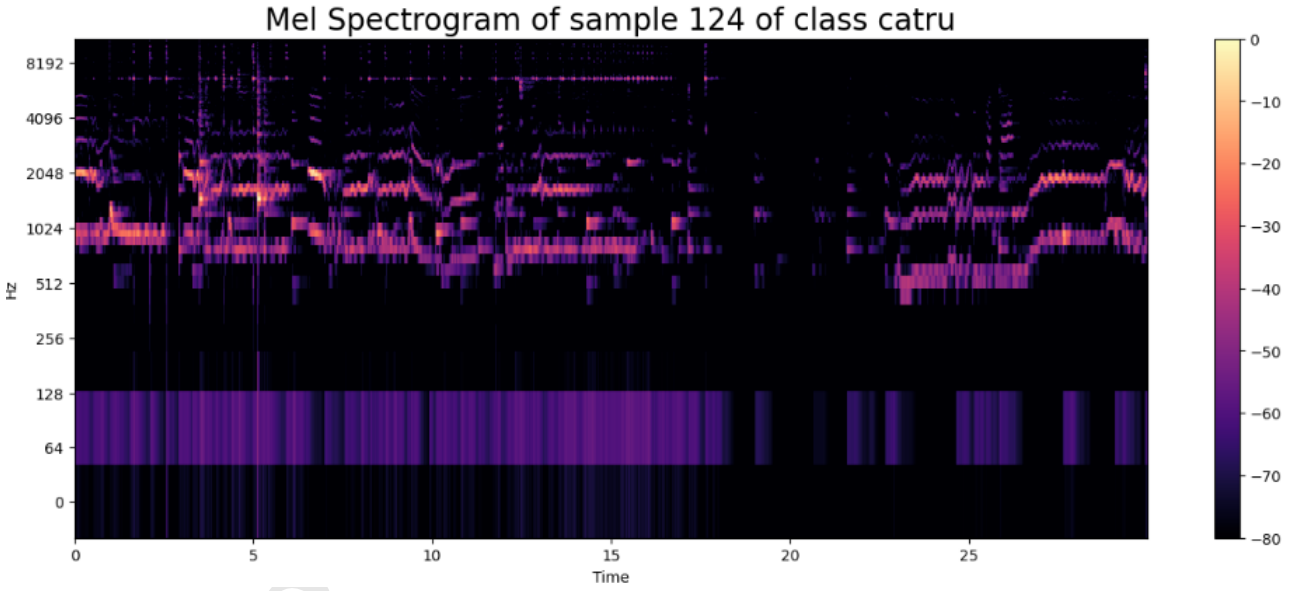

Figure 22: Mel-spectrogram of a sample

After extracting mel-spectrogram, we save mel-spectrogram images of 5 classes into the dataset folder via Matplotlib.pyplot API to construct images dataset for CNN classifiers. Each image has the shape of (126, 1290, 3).

3.3 Model building and Evaluation

3.3.1 Model building

(a) Train test split

Given the image dataset obtained from data processing stage, we separate this dataset into 3 subsets: Train, validation and test which ratios equal to 0.75, 0.15, 0.10 respectively. Then we save samples of these 3 subsets at train, val and test folders for the purpose of conveniently loading data.

(b) Data Augmentation

We apply rescale, zoom_range, height_shift_range, width_shift_range. We do not use other methods such as flipping or rotating because audio data has a temporal structure, meaning the order of the samples matters. Flipping or rotating the order of audio samples would distort the audio signal and render it useless for analysis or modeling. Furthermore, humans are highly sensitive to the temporal order of audio stimuli. In speech, for example, the order of phonemes, words, and phrases is essential for comprehension. Flipping or rotating audio would significantly alter the information conveyed by the sound, leading to nonsensical results. There are also more efficient augmentation methods to consider further including Noise injection, Mix-up, Frequency masking, etc. In the experiment, we feed both augmented data and non-augmented data to evaluate the effect of data augmentation on model performances. By experiment, training with augmented data requires more time and complex computation. Because of the limit of GPUs and time, we only experiment on training both types of data for model 1 to evaluate which type of data is better. Further experiments for augmented data will be conducted soon.

(c) Model Artchitecture

We propose 3 different CNN classifiers shown as below:

- Model 1: 4 convolutional layers, 1 dropout, 3 dense layers.
- Model 2: 3 convolutional layers, 1 dropout, 2 dense layers.
- Model 3: 3 convolutional layers, 2 dropouts, 3 dense layers, 2 batch norm layers.

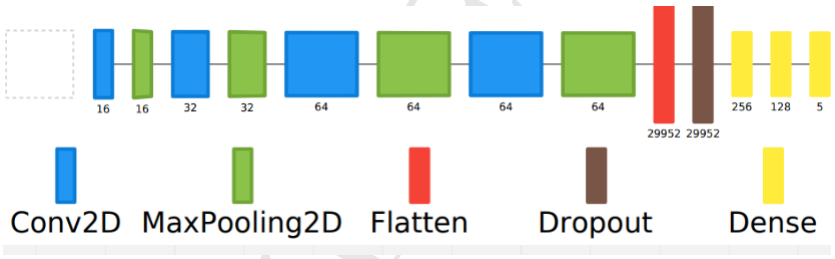

Figure 23: Model 1

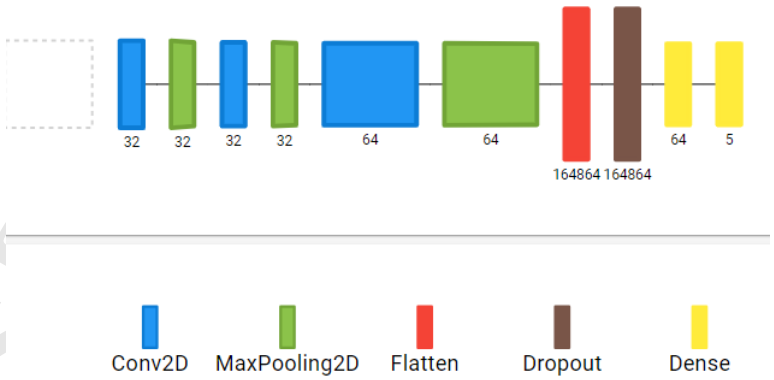

Figure 24: Model 2

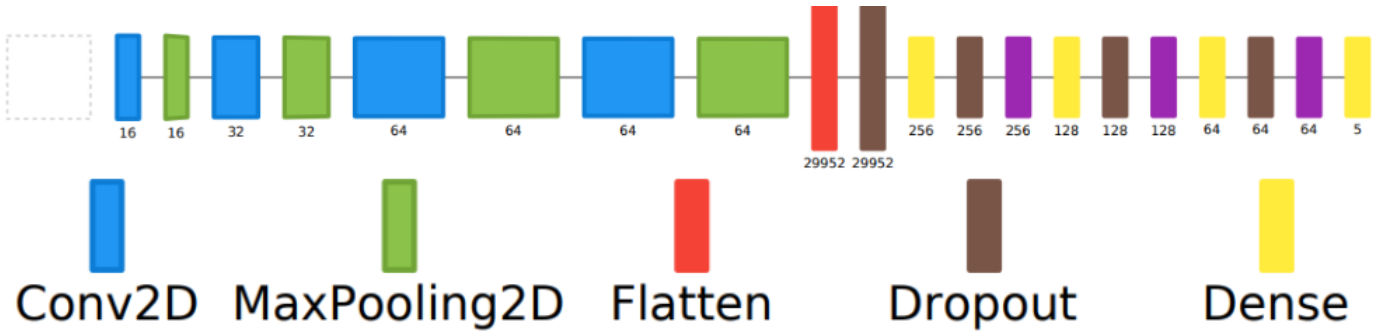

Figure 25: Model 3

(d) Model configuration

In the experiment on three models, we config model compile with the same arguments:

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

Figure 26: Compile model configuration

In the evaluation phase, we defined two more metrics:

```
model.compile(loss='categorical_crossentropy',                optimizer='adam',
metrics=["accuracy",
                    tf.keras.metrics.Precision(),  tf.keras.metrics.Recall()])
```

Figure 27: Define two more metrics

We define two callback features:

- EarlyStopping (to monitor the performance of the model during training and stop the training process when the performance on a validation set starts to degrade).

```
-   # Define callbacks
-   early = EarlyStopping(monitor='loss',
-       patience= 5,
-       verbose= 1,
-       mode='auto',
-       baseline= None,
-       restore_best_weights= True)
-
```

Figure 28: Early stopping

- Checkpoint (allows to save the model's weights during or after training and to save the best model so far, monitor the model's performance, or create checkpoints to resume training later). Here is an example of checkpoint config for model 1:

```
-   checkpoint1= tf.keras.callbacks.ModelCheckpoint(
-               filepath=     checkpoint_filepath     +     '/model1'     +
    '/model1_{epoch:02d}_{val_accuracy:.4f}.h5',
-       monitor='val_accuracy',
-       save_best_only=True,
-       save_weights_only=True,
-       verbose=1
-       )
```

Figure 29: Checkpoint

We train and evaluate three model consequently with epoch = 20, batch_size = 32, validation_batch_size = 32 on both augmented data and non-augmented data.

3.3.2 Evaluation

For this audio classification task, we evaluate the performance of three models via Loss, Accuracy, Precision, Recall and Confusion matrix.

(a) Model 1
This table below shows the performance of model 1:

	Non-augmented data	Augmented data
Loss	0,4680	0,7499
Accuracy	0,8960	0,8213
Precision	0,8954	0,8489
Recall	0,8907	0,8240

Table 1: Evaluation model 1 on validation set

	Non-augmented data	Augmented data
Loss	0,4514	0,3793
Accuracy	0,8840	0,8640
Precision	0,8984	0,8699
Recall	0,8840	0,8560

Table 2: Evaluation model 1 on test set

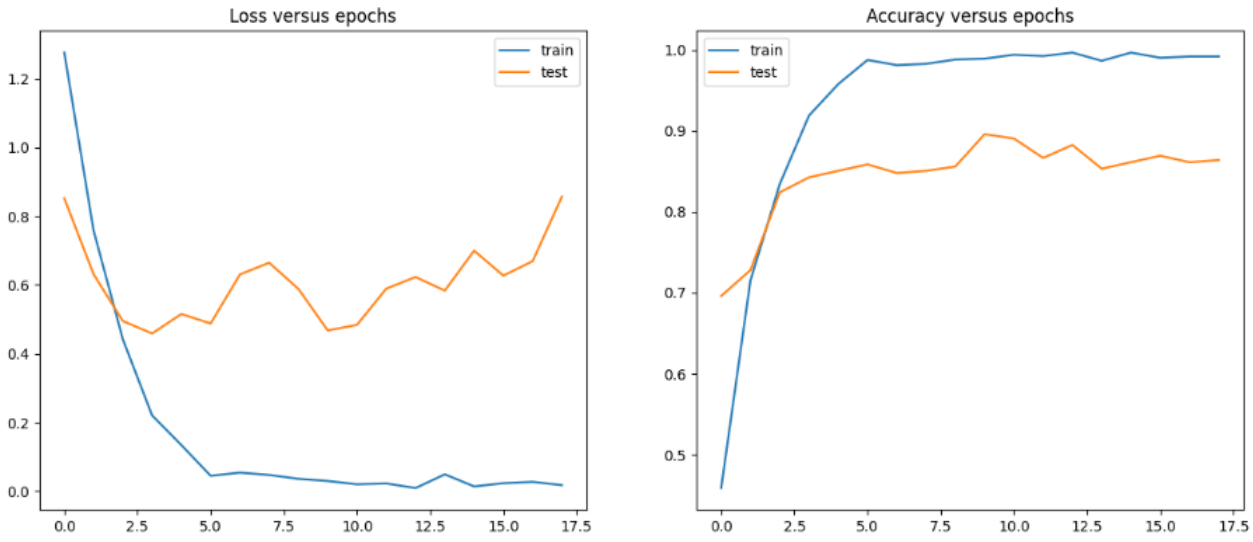

Figure 30: Loss and accuracy of model 1 (non-augmented data)

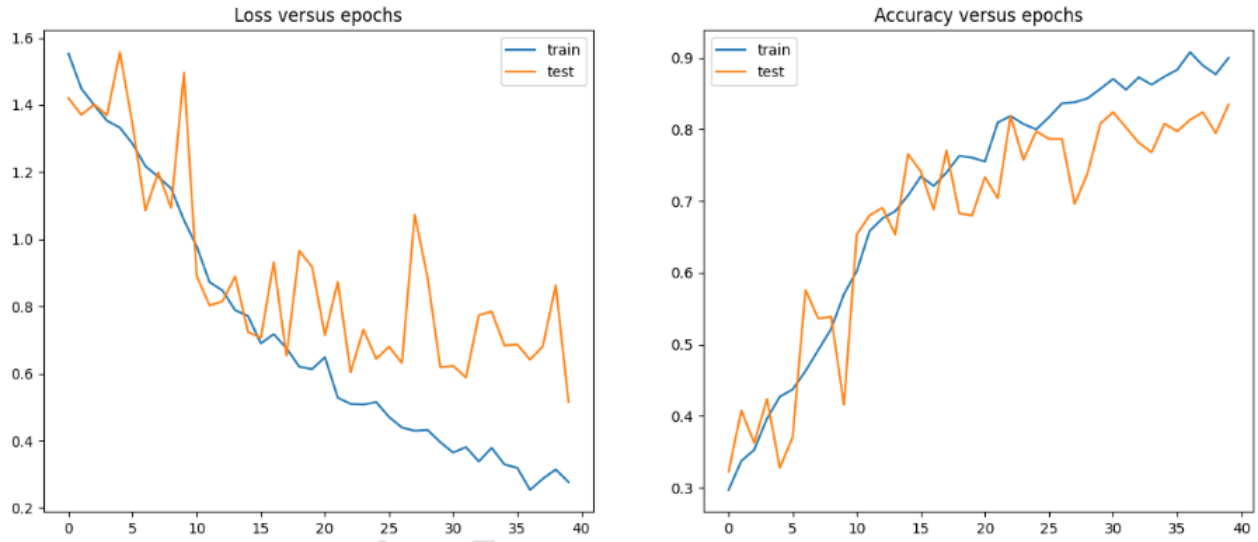

Figure 31: Loss and accuracy of model 1 (augmented data)

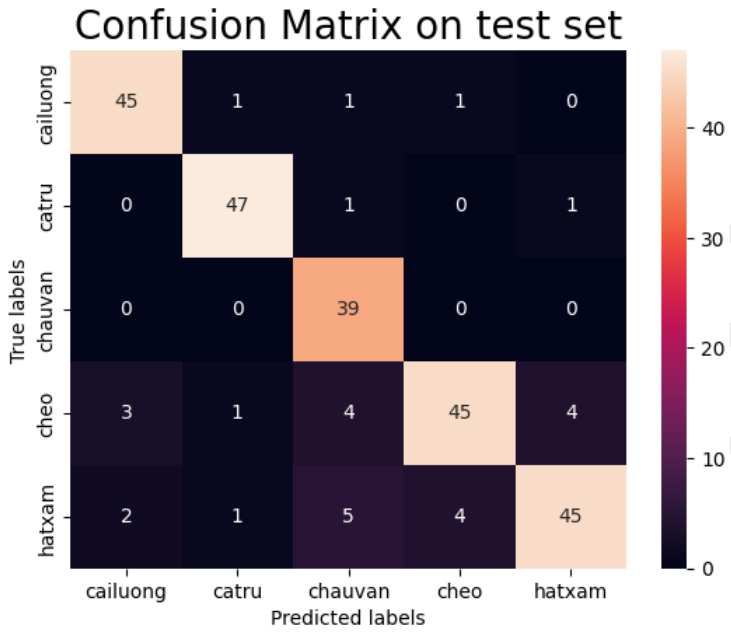

Figure 32: Confusion matrix model 1 on test set (non-augment data)

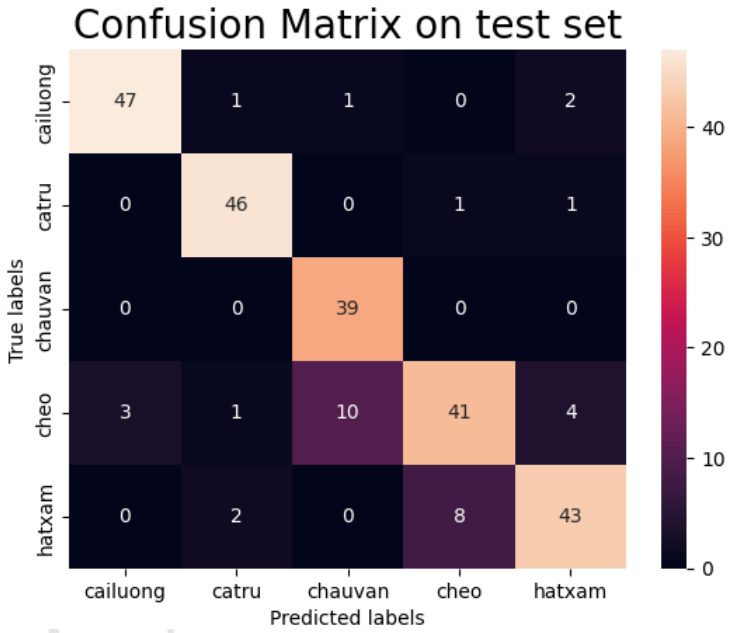

Figure 33: Confusion matrix model 1 on test set (augment data)

We can notice that the loss curve of Figure 31 tends to increase when we increase the number of training epochs. Furthermore, despite the acceptable accuracy on training set and test set, the difference between accuracy on training set and validation set is significant. These patterns indicate that an overfitting issue occurs in model 1 (with non-augmented data).

For augmented data, although the difference between accuracy on training set and validation set is reduced significantly and the loss curve trajectory tends to decrease. But there is a problem that on validation, the loss curve and the accuracy curve fluctuate which indicates the experience of some instability or inconsistency. In addition, model 1 with augmented data performs worse than that which with non-augmented at almost evaluation metrics. Why is this case? Here are some reasons that we propose to explain this issue:

- Inappropriate Augmentation: If the data augmentation techniques used are not suitable for the specific problem or dataset. Our augmentation methods such as zoom_range, height_shift_range, etc ... can break the spectral structure of Spectrogram images, which may not be appropriate for improving model performance and generalization. To tackle this problem, we can apply other methods like Mix-up, Frequency Masking,

etc.

- Insufficient Model Training: The model might not have been trained for a sufficient number of epochs or with an appropriate learning rate schedule, hindering its ability to benefit from the augmented data.
- Randomness in Augmentation: Augmentation techniques that introduce randomness might lead to variations in training data across different epochs, making the training process unstable.

(b) Model 2

	Score
Loss	0,6250
Accuracy	0,8720
Precision	0,8767
Recall	0,8720

Table 3: Evaluation model 2 on validation set

	Score
Loss	0,4261
Accuracy	0,8720
Precision	0,9032
Recall	0,8960

Table 4: Evaluation model 2 on test set

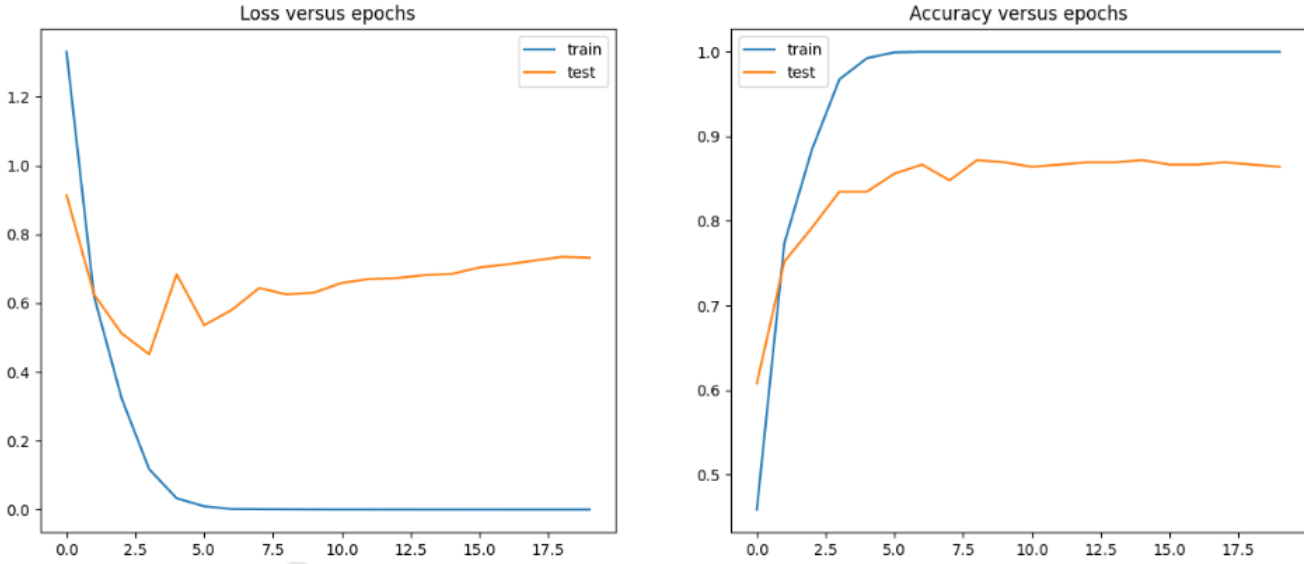

Figure 34: Loss and accuracy of model 2

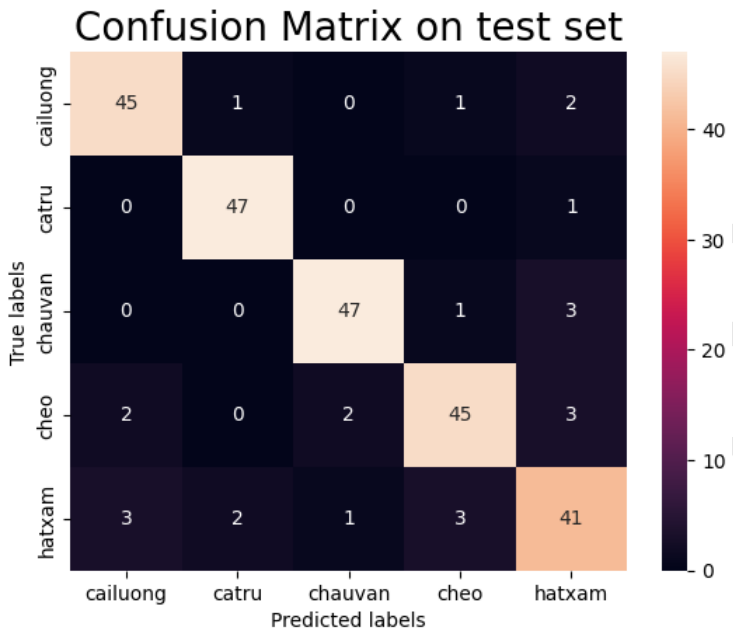

Figure 35: Confusion matrix model 2 on test set

Most of the evaluation of metrics in model 2 are improved clearly compared to those of model 1. Although the overall performance of model 2 is better than that of model 1, the signs of overfitting mentioned above still occur, which may result in poor generalization of the model when predicting new real-world data.

(c) Model 3

	Score
Loss	0,3558
Accuracy	0,9093
Precision	0,9212
Recall	0,9040

Table 5: Evaluation model 3 on validation set

	Score
Loss	0,2625
Accuracy	0,9093
Precision	0,9093
Recall	0,9320

Table 6: Evaluation model 3 on test set

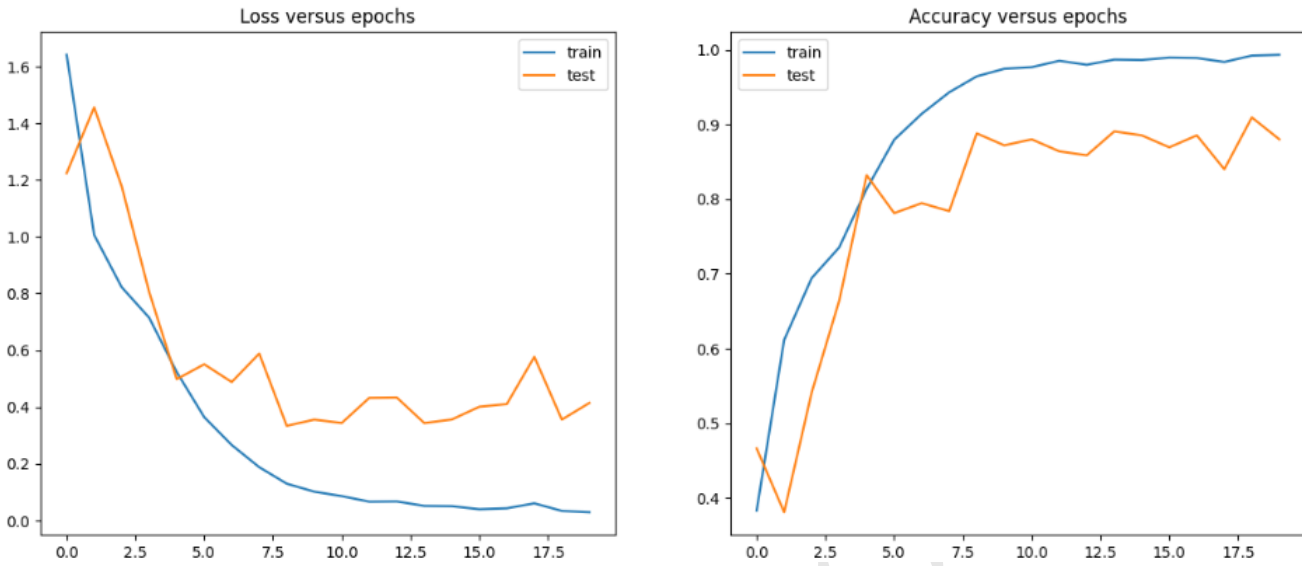

Figure 36: Loss and accuracy of model 3

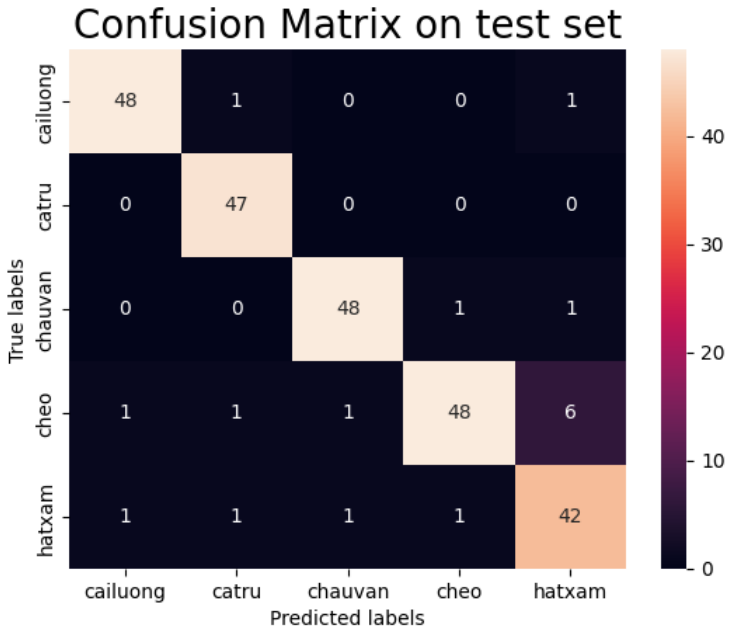

Figure 37: Confusion matrix model 3 on test set

By adding multiple Batch Norm layers, the instability and inconsistency problem in model 1 are somewhat improved, the loss curve tends to decrease to some extent, and the difference between accuracy on training set and test set are more acceptable. The overall performance of model 3 is better than those of both model 1 and model 2.

(d) Combine evaluation

	Model 1	Model 2	Model 3
Loss	0,4680	0,6250	0,3558
Accuracy	0,8960	0,8720	0,9093
Precision	0,8954	0,8767	0,9212
Recall	0,8907	0,8720	0,9040

Table 7: Evaluation model 2 on validation set

	Model 1	Model 2	Model 3
Loss	0,4514	0,4261	0,2625
Accuracy	0,8840	0,8720	0,9093
Precision	0,8984	0,9032	0,9357
Recall	0,8840	0,8960	0,9320

Table 8: Performance of three models on test set

Ensemble neural networks, also known as neural network ensembles, refer to a technique where multiple neural networks are combined to improve overall performance and enhance generalization. Ensemble techniques help with model selection by combining multiple models. Because each model is discover different aspects of the data and learns different features and representations of the data. By combining them, the ensemble can capture a more comprehensive understanding of the underlying patterns in the data the ensemble can provide a more balanced and robust decision.

In the inference phase, we propose to use late fusion of probabilities, referred to as PROD fusion. Consider predicted probabilities of each model as P_2=$[p_{s1}, p_{s2}, ..., p_{sC}]$ where is the number of classes and the S^{th} out of network evaluated. The predicted probabilities after PROD fusion is obtained by:

$$P_{prod}=[p_1, p_2, p_c] \text{ where } P_i = \frac{1}{S}\prod_{s=1}^{S} P_{si} \text{ for } 1 \leq i \leq C$$

Finally, the predicted label $\hat{y}$ is determined by: $\hat{y} = \text{argmax}\big(P_{prod}\big) = \text{argmax}\big(p_1, p_2, ..., p_c\big)$

3.4 Deploying

After completing building and evaluating model, we deploy model on a web platform that allows users to upload new audio data and get prediction results.

About new input audio files from user, both .mp3 and.wav are accepted, and multiple input audio files are also accepted. The system receives input audio and performs an end-to-end process including data processing, extraction features, and inference phase. The neuron network classifiers are set to train audio data which length is about 30 seconds. Hence, when receiving audio with a length greater than 30 seconds, this audio is split into equal 30s segment samples, then they will be fed into classifiers, and the final label of that audio is determined by voting among segment samples.

The prediction results will be displayed on the website and parallelly saved in the excel file in the system for recoding purposes.

Figure 38: Web platform main page

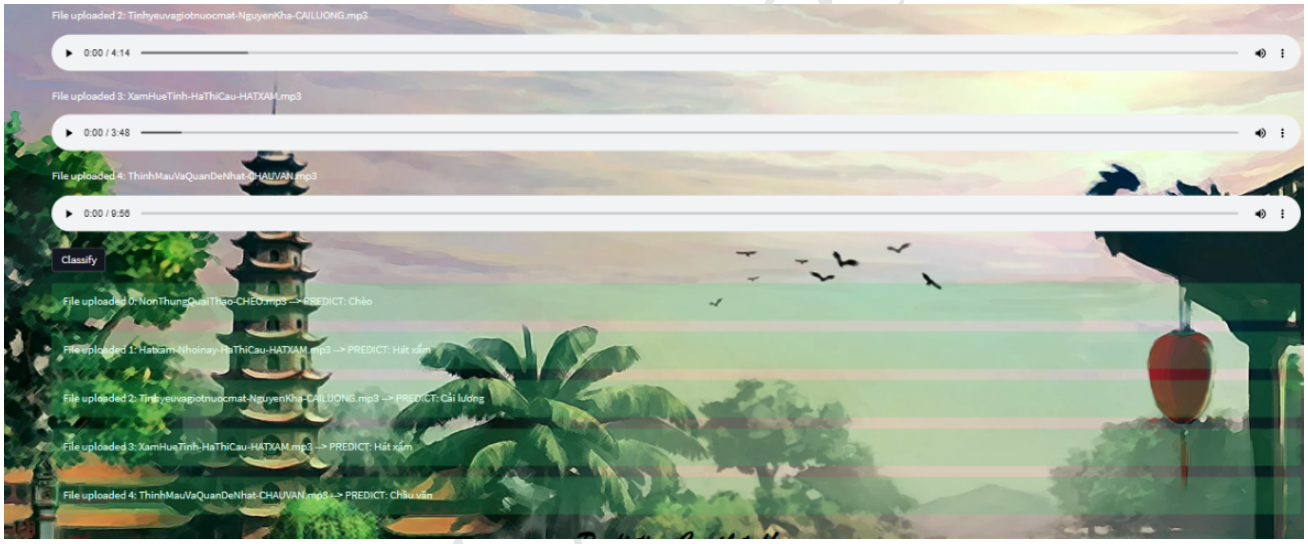

Figure 39: Get prediction results

A1

	A	B	C	D	E	F	G
1		Filename	Genre				
2	0	123.mp3	Chèo				
3	1	123.mp3	Hát xẩm				
4	2	234.wav	Chèo				
5	3	XamHueTinh-HaThiCau-HATXAM.mp3	Hát xẩm				
6	4	XamHueTinh-HaThiCau-HATXAM.mp3	Hát xẩm				
7	5	Hatxam-Nhoinay-HaThiCau-HATXAM.mp3	Hát xẩm				
8	6	QuanDeNhat-VanChuong-2815994.mp3	Chầu văn				
9	7	NonThungQuaiThao-CHEO.mp3	Chèo				
10	8	ThinhMauVaQuanDeNhat-CHAUVAN.mp3	Chầu văn				
11	9	TuongTienTuu-QuachThiHo-CATRU.wav	Ca trù				
12	10	Tinhyeuvagiotnuocmat-NguyenKha-CAILUONG.mp3	Cải lương				
13	11	ThinhMauVaQuanDeNhat-CHAUVAN.mp3	Chầu văn				
14	12	TuongTienTuu-QuachThiHo-CATRU.wav	Ca trù				
15							
16							
17							

Figure 40: Save prediction results

4 CHAPTER 4: EVALUATION AND DEVELOPMENT

In this project, we created an end-to-end system to solve an audio classification problem on a specific dataset through consequent stages, ranging from data processing, model building and evaluating, ensembling, and deploying. Our system achieves several goals such as:

- Effectively apply audio data processing methods, and build classification models with good accuracy.
- Gain insights into the impact of different factors affecting the performance of a classification model.
- Create an interactive interface between the user and the system, minimizing the complexity of exploiting a Machine Learning or Deep Learning model.

On the other hand, there are still some existing issues that need to be considered and resolved:

- **The proposed dataset:** This dataset is relatively clean and is also recorded in a relatively ideal environment with little noise, which is different from the actual dataset. Hence, that is why we achieve good results with only some data processing techniques and quite simple CNN architectures. It leads to the fact that our classifier can have poor performance and generalization when embedded into real-world systems. In actual life, datasets can be collected in non-ideal environments, where there are a lot of background noise and other unexpected patterns, which can be unpredictable and vary significantly across different samples. Hence more different challenges can be posed for effective data processing when requiring more techniques not only to remove unwanted noise but also to extract more valuable features to create more robust classifiers.
- **Lacking different feature extraction:** There are many other different features representing various aspects of audio data needed to consider. In this project, we are only interested in Time-Frequency Domain features such as Spectrogram, Mel-Spectrogram. To achieve good classification models with complex datasets, it is necessary to combine the extraction of many features to obtain more information about the audio data.
- **Ineffective data augmentation:** We failed to improve model performance and avoid overfitting through data augmentation because of an insufficient understanding of specific characteristics of data. Thus, we need to thoughtfully exploit our data so that we can apply appropriate data augmentation techniques to improve model performance.
- **Lack of diversity in model building:** We only do experiments on changing, adding, or deleting some layers between model architectures, resulting in not getting much significant breakthrough in model quality improvement.

Based on the existing problems mentioned above, we propose development directions to improve audio classification models with more complex datasets:

- Understanding different features and their applications for specific data to extract the most comprehensive features.
- Applying the same type of features while converting audios into images so that the model learns more aspects from each of them. For example, concatenate Spectrogram, Gamma Filter Spectrogram, and Constant-Q Transform into a 3D tensor for classification.
- Considering appropriate data augmentation techniques for audio data such as Mix-up, Frequency masking, Time shifting, etc.
- Building various CNN architectures for the sake of better ensemble.

References

[1] Saranga-K-Mahanta-google, “Audio Feature Extraction”, https://devopedia.org/audio-feature-extraction

[2] Wikipedia, “Spectrogram”. https://en.wikipedia.org/wiki/Spectrogram

[3] Wikipedia, “Convolutional Neuron Network”, https://en.wikipedia.org/wiki/Convolutional_neural_network

[4] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, https://arxiv.org/abs/1502.03167

[5] Raúl Gómez, “Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names”, https://gombru.github.io/2018/05/23/cross_entropy_loss/

[6] Evidently AI Team, “Accuracy, precision, and recall in multi-class classification”, https://www.evidentlyai.com/classification-metrics/multi-class-metrics

[7] Ho Manh Thang, “Vietnam Traditional Music (5 genres)”, https://www.kaggle.com/datasets/homata123/vntm-for-building-model-5-genres

[8] Librosa Library, https://librosa.org/

Link source code for this project is here: https://github.com/LTPhat/Vietnamese-Traditional-Music-Classification