TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A

PROJECT REPORT

ON

MUSIC RECOGNITION USING DEEP LEARNING

SUBMITTED BY:

PRADEEP KUMAR KHANAL(PUL075BEI002)

ACHYUT KAYASTHA (PUL075BEI005)

ASHISH KHATAKHO(PUL075BEI009)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

April, 2022

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **"Music recognition using deep learning"** submitted by **Pradeep Kumar Khanal**, **Achyut Kayastha**, **Ashish Khatakho** in partial fulfillment of the requirements for the Bachelor's degree in Electronics & Computer Engineering.

.....
Supervisor
Sanjivan Satyal
Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

.....
Internal examiner
Person B
Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

.....
External examiner
Person C
Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

Date of approval:

Copyright

Acknowledgement

We would like to express our sincere gratitude to all those who have contributed to the successful completion of our project, "Music Recognition Using Deep Learning".

First and foremost, we would like to thank our project supervisor, Asst. Prof. Sanjivan Satyal, for his valuable guidance, support, and insights throughout the project. His expertise in the field of signal processing, music, etc has been invaluable in shaping our ideas and helping us stay on track.

We would also like to express our gratitude to the faculty members and staff of Pulchowk Campus(Department of Electronics and Computer Engineering), who provided us with access to the necessary resources and facilities for our project. Their support and encouragement have been a constant source of motivation for us.

We are also grateful to our family and friends for their unwavering support and encouragement throughout our academic journey. Their love and belief in us have been a constant source of inspiration, and we are grateful for their presence in our lives.

Finally, we would like to express our appreciation to all those who have directly or indirectly contributed to our project. We have gained valuable insights and experiences from working on this project, and we hope that our work will be of benefit to others in the field of music recognition and deep learning.

Pradeep Kumar Khanal (PUL075BEI002)
Achyut Kayastha (PUL075BEI005)
Ashish Khatakho (PUL075BEI009)

Abstract

In our daily lives, we often listen to songs that we like and enjoy. However, there may be instances when we are in transit or at venues such as clubs or restaurants, and we hear a song playing in the background that catches our attention. We may desire to listen to this song again at a later time, but unfortunately, we may not be aware of the title of the song. As a result, we are unable to locate and listen to it again. Our project, titled "Music Recognition Using Deep Learning," aims to provide a convenient solution for identifying songs that are heard in various locations. While there are several existing popular applications, such as Shazam, SoundHound, and Google Sound Search, which offer music recognition services, we conducted an in-depth study of papers related to these apps to identify appropriate technologies and algorithms for our project.

Our project employs a deep neural network that leverages a contrastive learning approach for the purpose of song recognition. Initially, a large collection of songs is gathered and subjected to signal processing techniques, including Short Time Fourier Transform (STFT), mel filter bank, and decibel scale to generate log mel-spectrograms. These log mel-spectrograms are then fed into the neural network, which is trained to generate a fingerprint for each song at the segment level. These fingerprints are stored in a database.

In order to facilitate user access to our music recognition system, our model is integrated with a mobile app that features an interactive interface. When the user desires to identify a song, the microphone on their mobile device captures a brief audio sample, which is then sent to the server for fingerprint generation. This generated fingerprint is subsequently compared to the database of fingerprints to identify a similar match. Upon identification of a match, the app displays the corresponding title and URL of the identified song.

Keywords:*music recognition, mel-spectrograms, deep learning in music, audio fingerprint, audio augmentation, contrastive learning*

Contents

List of Figures

List of Tables

List of Abbreviations

STFT	Short Time Fourier Transform
URL	Uniform Resource Locator
FAISS	Facebook AI Similarity Search
UI	User Interface
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
CPU	Central Processing Unit
IDE	Integrated Development Environment
DFT	Discrete Fourier Transform
ELU	Exponential Linear Unit
ReLU	Rectified Linear Unit
JSON	JavaScript Object Notation
IVFPQ	Inverted File with Product Quantization
SNR	Signal-to-Noise Ratio

1. Introduction

1.1 Background

In modern times, the music industry has experienced significant growth and has become an integral aspect of many individuals' daily lives. Music enthusiasts listen to an array of songs, created by various artists, regularly. While it is easy to find songs of our liking through online searches, discovering new songs in public places poses a different challenge. Hearing a song for the first time in a public location makes it challenging to identify the title of the song, making it impossible to listen to it again. For example, one might hear a song being played in the background while waiting at a bus station, but without any knowledge of the title or artist, it becomes impossible to locate the song again. What would happen if there would be such an app that can capture snippets of the song being played and return the name of the song along with the URL? It would really be great, therefore we develop this project that recognizes the short snippet of audio being played.

There are several music recognition applications available in the market, such as Shazam[1], SoundHound[2], and Google Sound Search[3], that are capable of identifying songs. Each application employs a unique algorithm for song recognition. In our project, we evaluated various algorithms used in the field of music recognition and identified the most suitable one for our application. The system-building process begins with converting the audio into the desired format using signal processing techniques. Next, the audio is transformed into a log mel-spectrogram, which is a representation of the time, frequency, and loudness of sound using a Short Time Fourier Transform (STFT), a mel filter bank, and a dB scale conversion. The log mel-spectrogram serves as the primary feature of the audio. To improve the model's accuracy, augmented versions of the original audio are created by incorporating background noise, IR impulse response, and other techniques. The log mel-spectrogram is generated for each augmented audio clip, and both the original and augmented versions are used for neural network training. A contrastive learning approach is utilized during the training process to ensure that the embeddings of the original songs and their augmented versions are close, while the embeddings of other songs are further away. The model is optimized by minimizing the loss function in each epoch. Once the model is trained, it is used to generate fingerprints for each song, which are stored in a database. These fingerprints are compared to the fingerprint of the input audio snippet using FAISS technology.

To make the system accessible to users, the model is integrated into a mobile app that

provides an interactive interface. The mobile microphone is used to capture the audio, and the trained model is used to generate a fingerprint of captured audio. The fingerprint is compared to the fingerprints in the database, and the name, URL and other info of the matching song are returned to the user.

1.2 Problem Statement

Music enthusiasts often encounter situations where they are unable to identify a song they hear in public places like cafes, shopping malls, bus stations, etc and would like to listen to it again. This is due to the lack of knowledge about the song's title, which makes it difficult to search for it online. In order to address this problem, it is necessary to develop a music recognition system that can identify short snippets of audio being played, capture the relevant audio, and return the song's name along with its URL. Therefore, the primary goal of our project is to design and develop an efficient and user-friendly music recognition system that can benefit music enthusiasts globally.

1.3 Objectives

The objectives of our project are listed below:

- To develop a robust music recognition system that can recognize/identify recorded clip of a song regardless of background noise, recording quality, and duration of the recorded clip
- To develop a mobile app and integrate the music recognition system into it

1.4 Applications

The applications that are possible from a music recognition system are listed below:

- It can be used to identify copyrighted music used on various social media platforms.
- It can be used in broadcast monitoring so that television and radio broadcasts couldn't use copyrighted music without giving proper royalties.
- It can be used in forensic investigations to identify audio used in criminal activities or to analyze audio evidence.
- It can also be used for research and study of audio, frequency, and other audio-related things.

2. Literature Review

2.1 Related Work

Music recognition systems are becoming increasingly popular as they offer an efficient and effective way to recognize songs or music tracks, especially in large music libraries. These systems use various techniques such as digital signal processing, machine learning, and pattern recognition to identify a song from its audio signal. This literature review aims to provide an overview of the current state of the art in music recognition systems.

Music recognition systems have been studied extensively for decades. The earliest music recognition systems were based on audio fingerprinting techniques, which involved extracting distinctive features from an audio signal and comparing them to a pre-existing database of songs. However, these systems were limited in their accuracy and were only able to recognize a limited number of songs.In recent years, music recognition systems have improved significantly, thanks to advances in machine learning and deep learning techniques. These systems are now able to recognize a wider range of songs with greater accuracy.

Music recognition systems use a variety of techniques to recognize songs, including audio fingerprinting, machine learning, and deep learning. Audio fingerprinting involves extracting distinctive features from an audio signal and comparing them to a pre-existing database of songs. This technique is effective in identifying a song even when the audio quality is poor or when there is background noise.

An audio fingerprint is most for the music recognition system and a fingerprint is the one that conveys the identity of a song like the fingerprint of a human conveys the identity of the human being. A fingerprint system generally consists of two components: a method to extract fingerprints and a method to efficiently search for matching fingerprints in a fingerprint database[4]. General requirements for audio fingerprinting systems are discriminability over a huge number of other fingerprints, robustness against various types of acoustic distortions, and computational efficiency for processing large-scale databases[5]. Using matching algorithms and fingerprints gnarled interpretation of recording can be identified as the same audio content and later stored in the database[6].

To date, much research had been carried out on audio fingerprinting and music recognition systems. We have also studied some of the papers to find the working of various algorithms and find the best one for our project. Below we have presented a summary of various studies:

The paper “A Highly Robust Audio Fingerprinting System” by Haitsma and Kalker

(2002) proposes a new audio fingerprinting system that is highly robust against various types of distortions and noise. It starts by introducing the concept of audio fingerprinting, which involves extracting a small set of features from an audio signal to identify the corresponding song. The authors note that previous audio fingerprinting systems were not robust enough to handle various types of distortions and noise. The proposed audio fingerprinting system consists of two main components: feature extraction and matching. In the feature extraction phase, the audio signal is divided into short frames, and a set of features, called perceptual hashes, are generated from each frame. The perceptual hashes are computed based on the properties of the human auditory system, such as the critical band filtering and the masking effect. In the matching phase, the perceptual hashes are compared to a pre-existing database of hashes to identify the corresponding song.The authors demonstrate the robustness of their system against various types of distortions and noise, such as time stretching, pitch shifting, additive noise,etc.

The paper "An Industrial-Srength Audio Search Algorithm" by Wang (2003) reveals audio searching algorithm of the popular app Shazam.. The algorithm is noise and distortion resistant, computationally efficient, and massively scalable, capable of quickly identifying a short segment of music captured through a cellphone microphone in the presence of foreground voices and other dominant noise, and through voice codec compression, out of a database of over a million tracks[7]. The algorithm uses a combinatorially hashed time-frequency constellation analysis of the audio, yielding unusual properties such as transparency, in which multiple tracks mixed together may each be identified. It takes the frequency. The recognition rate was demonstrated for the various value of the SNR.

The paper "Now Playing:Continuous low-power music recognition" (2017) presents a low-power music recognizer that runs entirely on a mobile device and automatically recognizes music without user interaction [8]. To reduce battery consumption, a small music detector runs continuously on the mobile device's digital signal processor (DSP) chip and wakes up the main application processor only when it is confident that music is present. Once woken, the recognizer on the application processor is provided with a few seconds of audio which is fingerprinted and compared to the stored fingerprints in the on-device fingerprint database of tens of thousands of songs. It takes the mel-spectrogram as the input for the neural network and generates the embeddings for the audio content.It performs a sequence search in two stages to find the matching fingerprint from the database. It shows the evaluation of the performance of the NNFP and the matching algorithm using 64, 96, and 128-dimensional fingerprints on a set of 20k 8s long audio segments from 10k different songs. The 96 and 128-dimensional models clearly outperform the 64-dimensional model with the 128-dimensional model being only slightly better than the 96-dimensional model.

The paper, "Neural Audio Fingerprint for High-specific Audio Retrieval based on Contrastive Learning" (2020) releases the algorithm of neural audio fingerprinting which is based on contrastive learning. It takes the log-mel spectrogram as the input to the neural network. Various augmentation tasks are done here to make the system more robust and contrastive learning makes use of augmented audio. It also compares their model with the Now playing[8] and Dejavu open source project and found their system performed well. It also made a study on the hit rate on various query lengths, dimension embedding, and batch sizes. It concluded that the longer the query duration more is the hit rate. Also larger the batch size, the better performance was seen in all experiments. It also finds when the dimension of embedding was changed from 64 to 128, the performance raised up to 7.6%[5].

2.2 Related Theory

2.2.1 Audio Analysis

Audio analysis is the process of analyzing an audio signal to extract useful information from it. It involves breaking down the audio signal into its component parts and analyzing them separately. One commonly used method for audio analysis is the creation of spectrograms. A spectrogram is a visual representation of the frequency spectrum of an audio signal over time. It shows how the frequency content of the signal changes over time, which can provide useful information about the characteristics of the audio signal.

When we listen to the sound, our ears perceive sound in a logarithmic manner, which means that we perceive differences in frequency in a nonlinear way. For example, the difference between 100 Hz and 200 Hz may seem smaller to us than the difference between 10,000 Hz and 10,100 Hz, even though both differences are 100 Hz. To account for this nonlinear perception of sound, a scale called the Mel scale was developed. The Mel scale is a logarithmic scale that is based on the way our ears perceive sound. By using the Mel scale, we can more accurately represent the frequency content of an audio signal in a way that is consistent with how our ears perceive sound.

To create a Mel spectrogram from a time signal, we follow these steps:

Preprocessing: The first step is to preprocess the audio signal. This can include steps such as resampling the signal to a fixed sampling rate, applying a window function to the signal to reduce spectral leakage, and removing any DC offset or low-frequency noise.

Frame the signal: Next, the signal is divided into small segments or frames, usually around 20-30 milliseconds in duration. Each frame is then processed separately.

Fourier Transform: For each frame, a Fourier transform is applied to the signal to obtain the power spectrum. The power spectrum represents the amount of energy in each frequency

band.

Apply Mel filterbank: Next, a filterbank is applied to the power spectrum to obtain a set of Mel-filtered power spectra. The filterbank is designed to mimic the way the human ear perceives sound, by emphasizing certain frequency bands and de-emphasizing others. This is achieved by using a set of triangular filters spaced evenly on the Mel scale.

Logarithmic scaling: The Mel-filtered power spectra are then converted to a logarithmic scale to account for the logarithmic perception of sound by the human ear. This is typically done using the logarithm base 10 or the natural logarithm.

Spectrogram: The final step is to plot the Mel spectrogram, which is a visual representation of the Mel-filtered power spectra over time. The Mel spectrogram is usually displayed as a two-dimensional heat map, where the x-axis represents time and the y-axis represents frequency.

Mel spectrograms are a powerful tool for analyzing audio signals, as they provide a more accurate representation of the frequency content of the signal compared to traditional spectrograms. They are commonly used in applications such as speech recognition, music analysis, and sound classification.

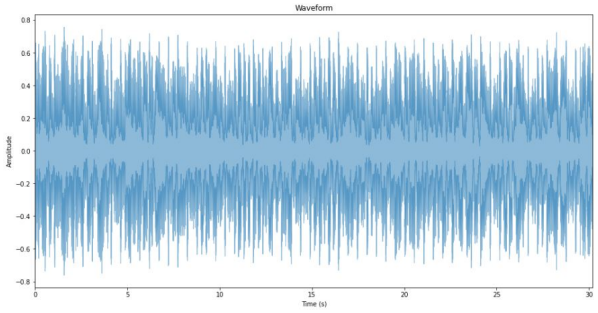


Figure 2.1: Audio signal in Time domain

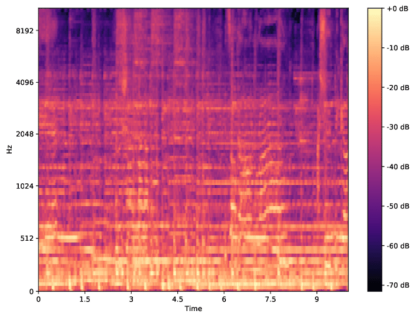


Figure 2.2: Mel-spectogram

2.2.2 Contrastive Learning

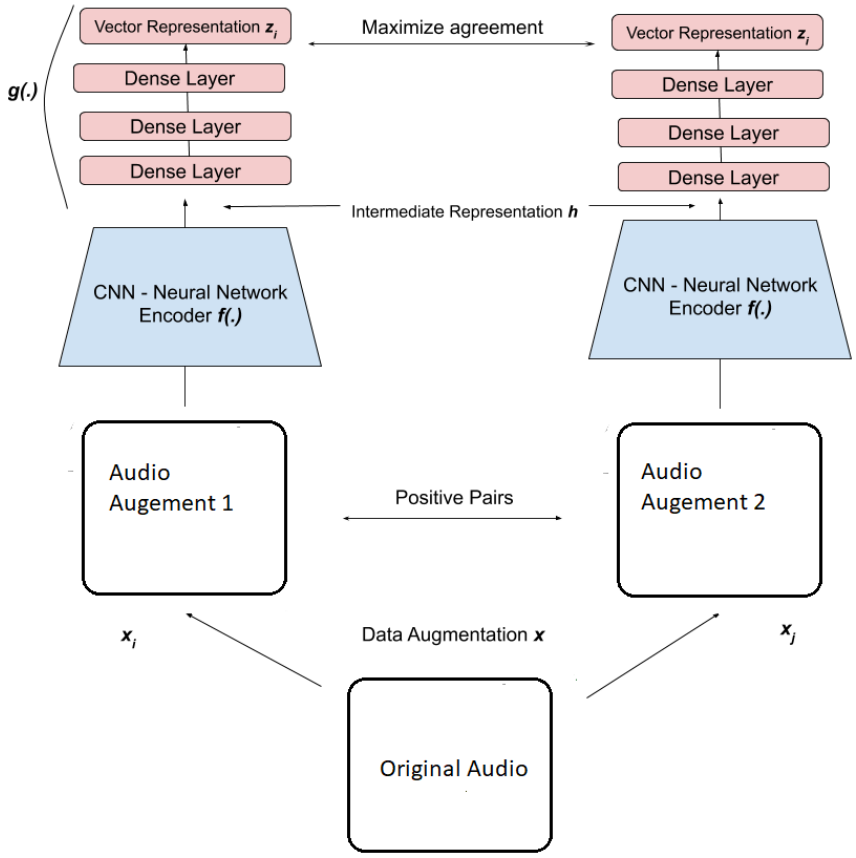


Figure 2.3: Contrastive Learning Framework

Contrastive learning is a type of unsupervised learning that is used in audio fingerprinting to identify audio signals based on their unique acoustic characteristics. In contrastive learning, the goal is to learn a representation of the input data that maximizes the difference between positive and negative examples.

In audio fingerprinting, contrastive learning is used to learn a feature representation of an audio signal that captures its unique acoustic characteristics. This is done by training a deep neural network on pairs of audio signals, where one signal is a positive example (i.e., a recording of the same song) and the other signal is a negative example (i.e., a recording of a different song).

During training, the network is trained to maximize the similarity between positive examples while minimizing the similarity between negative examples. This is typically done using a loss function such as the contrastive loss or triplet loss, which encourages the network to learn a feature representation that clusters similar audio signals together and separates dissimilar audio signals.

Once the network has been trained, it can be used to generate a compact and discriminative representation of an audio signal, known as an audio fingerprint. Audio fingerprints can be used to identify an audio signal by comparing it to a database of pre-computed fingerprints. This is typically done using a similarity metric such as the cosine similarity or Euclidean distance.

Audio fingerprinting using contrastive learning has many applications, including music identification, audio classification, and content-based retrieval. It is a powerful technique that can be used to identify audio signals even in the presence of noise, distortion, or other types of audio degradation.

2.2.2.1 Convolutional encoder

In contrastive learning, convolutional encoders are used to learn useful feature representations from image data in a self-supervised manner. A convolutional encoder is a type of neural network architecture that consists of several convolutional layers followed by one or more fully connected layers.

During training, the convolutional encoder is fed pairs of images, and the objective is to learn a feature representation that maps similar images to nearby points in the embedding space and dissimilar images to distant points. This is achieved through the use of a contrastive loss function, which penalizes the distance between similar pairs and encourages the distance between dissimilar pairs to be greater than a certain margin.

The convolutional encoder is typically pretrained on a large dataset of unlabeled images, such as ImageNet, using the contrastive loss function. The resulting feature representation can then be used in downstream tasks, such as image classification or object detection, by fine-tuning the pretrained encoder on a smaller labeled dataset.

One advantage of using convolutional encoders in contrastive learning is that they can learn useful features from raw image data without the need for explicit labels. This makes it possible to leverage large amounts of unlabeled data to learn representations that can generalize well to new tasks and datasets.

Another advantage of using convolutional encoders in contrastive learning is that they are well-suited to capturing local features in images, such as edges and textures. This is because the convolutional layers apply a set of learnable filters to the input image, which extract local features at different scales. The resulting feature maps are then aggregated by the fully connected layers to produce a global feature representation that can be used in downstream tasks.

Overall, convolutional encoders are a powerful tool for learning useful feature representations from image data in a self-supervised manner, which can then be used in downstream

tasks through fine-tuning. They have been widely used in contrastive learning to learn representations that are useful for a wide range of computer vision tasks.

2.2.2.2 Divide and Encode Layer

The Divide and Encode layer is a specific layer used in contrastive learning models for audio and speech processing. It is designed to split the input audio signal into multiple segments or frames, which are then individually encoded and processed by the network.

The Divide and Encode layer consists of two main components: the Divide component and the Encode component. The Divide component is responsible for dividing the input audio signal into smaller segments or frames. These segments are typically non-overlapping and may have a fixed or variable length. The Encode component is responsible for encoding each segment or frame of the audio signal using a convolutional neural network (CNN) or other encoding scheme.

After the Divide and Encode layer, the encoded representations of each segment or frame are combined or aggregated to produce a final embedding vector that represents the entire audio signal. This final embedding vector is then used for downstream tasks such as audio classification or speech recognition.

Overall, the Divide and Encode layer is an important component of contrastive learning models for audio and speech processing because it enables the network to learn representations that capture the temporal dynamics and structure of audio signals.

2.2.2.3 NTxent

NTXent (or NT-Xent) loss is a loss function used in unsupervised learning of neural network-based representations (also known as embeddings) for high-dimensional data, such as images or text. The term "NTXent" stands for "Normalized Temperature-scaled Cross-Entropy". The NTXent loss is designed to maximize the similarity between pairs of samples from the same class (known as "positive pairs") while minimizing the similarity between pairs of samples from different classes (known as "negative pairs").

The NTXent loss function is computed by first normalizing the dot product of the embeddings with a temperature parameter and applying the softmax function. The resulting probabilities are used to compute the cross-entropy loss between the embeddings of a sample and its positive and negative pairs. The NTXent loss has been shown to be effective in a variety of applications, including image and text retrieval, and has been used in several state-of-the-art models such as SimCLR (Simple Framework for Contrastive Learning of Visual

Representations)[9] and MoCo (Momentum Contrast).

$$\text{NT-Xent} = -\log \frac{\exp\left(\text{sim}(q,k)/\tau\right)}{\exp\left(\text{sim}(q,k)/\tau\right) + \sum_{k' \in \mathcal{K}_\neg(q)} \exp\left(\text{sim}(q,k')/\tau\right)} \tag{2.1}$$

where:

q is the query embedding k is the positive key embedding (i.e., a different view of the same data point) $k' \in \mathcal{K}_\neg(q)$ are the negative key embeddings (i.e., embeddings of other data points) $\text{sim}(a,b)$ is the cosine similarity between embeddings a and b, τ is a temperature parameter that scales the similarity values

2.2.3 Optimizer

In machine learning, an optimizer is an algorithm or method used to adjust the parameters of a model in order to minimize the loss function. The loss function is a measure of how well the model is performing on the training data, and the goal of the optimizer is to find the set of parameters that result in the lowest loss.

Optimizers work by iteratively adjusting the parameters of the model in a direction that reduces the loss. The adjustment of the parameters is guided by the gradient of the loss function with respect to the parameters. The gradient indicates the direction in which the loss is decreasing the fastest, so the optimizer adjusts the parameters in that direction.

There are many different optimizers that can be used in machine learning, including stochastic gradient descent (SGD), Adam, RMSprop, Adagrad, and Adadelta, among others. Each optimizer has its own characteristics and is suited for different types of problems.

2.2.3.1 ADAM Optimizer

The Adam optimizer is an optimization algorithm used in machine learning and deep learning for stochastic gradient descent. It was introduced in a 2014 paper titled "Adam: A Method for Stochastic Optimization" by Diederik P. Kingma and Jimmy Ba.

Adam combines ideas from two other optimization algorithms: Adagrad, which adapts the learning rate for each parameter based on the historical gradients, and RMSProp, which divides the learning rate by a running average of the squared gradients.

The key idea behind Adam is to use both first and second moments of the gradient to adapt the learning rate for each parameter. Specifically, Adam maintains two exponentially decaying averages of the gradient: the first moment (the mean) and the second moment (the uncentered variance). These averages are used to compute the learning rate for each parameter at each iteration.

In addition, Adam introduces bias-correction to these averages, which is necessary especially in the early stages of training when the number of iterations is small. The bias-

correction step involves dividing the first and second moments estimates by the corresponding decay rates. This is done to compensate for the fact that the averages are initialized at zero and therefore biased towards zero in the early stages of training.

Overall, the Adam optimizer is a popular choice for many deep learning tasks due to its effectiveness and ease of use. It has been shown to converge faster and achieve better performance than other optimization algorithms such as stochastic gradient descent and Adagrad.

2.2.3.2 Lamb Optimizer

LAMB (Layer-wise Adaptive Moments optimizer for Batch normalization) is an optimization algorithm used in deep learning that was proposed in 2019 by You et al. The LAMB optimizer combines two popular optimization techniques, Adaptive Moment Estimation (Adam) and Layer-wise Adaptive Scaling (LARS), to provide an effective and efficient optimization algorithm.

Like Adam, LAMB uses adaptive learning rates for each weight parameter in the network. This helps to ensure that each weight is updated with an appropriate learning rate based on its gradient. However, unlike Adam, LAMB also incorporates layer-wise normalization, which helps to ensure that the weight updates are appropriate for each layer of the network.

The key innovation of LAMB is the use of a trust ratio, which balances the effect of the adaptive learning rate and the layer-wise normalization. The trust ratio is a scalar value that is used to scale the learning rate for each weight. If the trust ratio is large, the adaptive learning rate dominates, and if it is small, the layer-wise normalization dominates.

In addition to the trust ratio, LAMB also uses an adaptive weight decay term, which helps to prevent overfitting by regularizing the weights of the network. The weight decay term is scaled by the same trust ratio as the learning rate, which helps to ensure that the regularization is appropriate for each weight.

Overall, LAMB is an effective optimization algorithm that combines the best features of Adam and LARS to provide fast and accurate training of deep neural networks. It has been shown to outperform other optimization algorithms such as SGD, Adam, and Adagrad on a variety of deep learning tasks.

2.2.4 Sequence search

Sequence search is the task of finding similar sequences in a large database of sequences. This is a common problem in many areas of science and engineering, including bioinformatics, natural language processing, and computer vision. One approach to solving the sequence search problem is to use a nearest neighbor search algorithm such as Faiss.

Faiss (Facebook AI Similarity Search) is a library for efficient similarity search and clustering of dense vectors. It was developed by Facebook AI Research and is widely used in industry and academia. Faiss provides several algorithms for approximate nearest neighbor search, including Product Quantization, Hierarchical Navigable Small World (HNSW), and Inverted File Search (IVF). These algorithms can be used to search large databases of vectors quickly and accurately, making them well-suited for sequence search applications.

To use Faiss for sequence search, we first need to represent each sequence as a vector. This can be done using a technique such as word embedding or sequence encoding. Once we have a vector representation of each sequence, we can build an index using one of the Faiss algorithms. The index allows us to perform fast approximate nearest neighbor search on the database of sequences.

During search, we provide a query sequence to the Faiss index, and it returns a set of nearest neighbor sequences in the database. The nearest neighbor sequences are those that have the closest vector representation to the query sequence. The distance metric used to compute the similarity between vectors can be chosen based on the application. For example, cosine similarity or Euclidean distance can be used.

IVFPQ "Inverted file with product quantization", which is an extension of the inverted file technique used for efficient similarity search and clustering of high-dimensional datasets.

IVFPQ is designed to overcome the limitations of the traditional inverted file approach by reducing the memory usage and query time. In IVFPQ, the dataset is first partitioned into Voronoi cells based on their distance from a set of centroids. Each Voronoi cell is then compressed using product quantization, which splits the high-dimensional vectors into smaller sub-vectors and quantizes each sub-vector separately. The quantized sub-vectors are then stored in a codebook, which is used to map the quantized sub-vectors back to their original high-dimensional space.

During search, the query vector is also compressed using product quantization and assigned to its nearest centroid in each sub-vector. The inverted file for each Voronoi cell is then searched using the compressed query to find the most similar data points.

IVFPQ has been shown to provide significant improvements in both memory usage and query time over the traditional inverted file approach. It is widely used in applications such

as image and text search, recommendation systems, and clustering.

Overall, sequence search using Faiss is a powerful technique that can be used to find similar sequences in large databases quickly and accurately. It has many applications, including DNA sequence alignment, natural language processing, and image retrieval.

The general workflow of FAISS involves the following steps:

- Data preparation: The first step is to prepare the data that is to be indexed by FAISS. This typically involves representing the data as a matrix of high-dimensional vectors, where each row represents a data point. The data can be stored in a variety of formats such as NumPy arrays, PyTorch tensors, or other custom data structures.

- Index creation: The next step is to create an index of the data using one of the indexing techniques provided by FAISS. The choice of indexing technique depends on the size and nature of the data as well as the performance requirements.

- Training (Optional): In some cases, such as with unsupervised clustering, FAISS may require training to create the index. In this case, the data is fed into the training process, which generates a set of parameters that define the index structure.

- Adding data: Once the index is created, data can be added to it incrementally. This is useful in cases where new data is added over time and needs to be included in the index.

- Search: Given a query vector, FAISS searches the index to find the vectors that are most similar to the query. This is done by calculating the cosine similarity or L2 distance between the query vector and the vectors in the index. The search results can be ranked by similarity score, and the top-k results can be returned.

- Batch search (Optional): FAISS supports batch search, which allows multiple query vectors to be searched at once, providing significant speedups for large datasets.

- Clustering: FAISS also supports clustering of the data, which involves grouping similar vectors together into clusters. Clustering can be useful for tasks such as anomaly detection or recommendation systems.

- GPU acceleration (Optional): FAISS can also be used with GPUs to achieve even faster search and clustering performance.

3. System analysis

3.1 Requirement Analysis

3.1.1 Functional Requirement

The functional requirements of our system are as follows:

1. The system must contain a microphone to record the audio sample.
2. The system must be able to convert the audio to its digital fingerprint
3. The system must be able to identify the actual song from a recorded audio sample(query)
4. The system should provide info on the resulting song like name, artist and URL.

3.1.2 Non-Functional Requirement

These requirements are not needed by the system for its completion but are essential for the better performance of the Music Recognition System. The points below focus on the non-functional requirement of the system.

1. **Reliability**: This system should be reliable and robust enough to handle various recorded input audios. We intend to make our system reliable as possible by performing various optimization and error-handling techniques.
2. **Maintainability**: The system will be modularized and divided into smaller sub modules as it develops. As a result, this system will be easy to maintain because each sub-module will be easy to check and manage.
3. **Availability**: The system can be accessed anywhere and anytime with a decent internet connection thus making the system usable.
4. **Accuracy**: The output song given by the system should be exact to the recorded input audio.
5. **Performance**: The music recognition system should provide fast and accurate results within a reasonable time frame.

6. **Portability**: The system we are developing will be portable over various mobile devices operating on Android OS and Apple's IOS.

3.1.3 Software Requirement

Software that are required for building a music recognition system are listed below:

1. Operating System: Windows 10
2. Python Programming Language: Version 3.x or higher
3. Python Libraries: Tensorflow, Keras, NumPy, Pandas, Matplotlib,
4. Video downloader: Yt-dlp
5. Multimedia file handler: FFmpeg
6. IDE: Visual Studio Code
7. Mobile app development framework: Flutter
8. Diagramming tool: Microsoft Visio
9. Google Colab
10. shell

3.1.4 Hardware Requirement

Hardware that are required for building a music recognition system are listed below:

1. Computer/Laptop with minimum intel core i5 or similar processor with 8GB RAM
2. GPU in case of fast training

3.2 Feasibility Study

The following points describe the feasibility of the project.

3.2.1 Economic Feasibility

The expenditure for this project is on computational power. The processing power of this system is mainly for the training of models which requires good GPU and memory. In addition to our laptops, We will be using google colaboratory or Kaggle notebook for training our model.Also, all the software we are using in the project is free of cost so, the project is economically feasible.

3.2.2 Technical Feasibility

Technically, this project is quite challenging as we'll be working on different frequencies and spectrograms, collecting datasets as much as possible for the effectiveness of the system. However, this project seems technically feasible as we are familiar with Python and are thinking to implement various neural models using python. And, for this project, we are using Windows as our main OS with an i5 processor. Technologies we use on this project are fully available due to which we don't have to face any problem of scarcity of tools so implementing this system seems to be technically feasible.

3.2.3 Operational Feasibility

This system requires recorded audio from the user which is then processed by the model to find the actual match of audio. The audio input can easily be recorded by using the mic from a mobile device via a mobile application for the system. Also, the computation is completely on the server side so the hardware requirements of the mobile devices won't be much of an issue. Users of the system will not require any expertise in a technical field to use it. So, the project is operationally feasible.

3.2.4 Time Feasibility

A certain time frame is designed to accomplish the project within in deadline. Also, each team member is assigned a particular task due to which the project can be delivered faster.

4. Methodology

4.1 Data collection

There is no availability of a dataset of Nepali songs which is a challenging issue. To overcome this, we manually compiled a playlist of Nepali songs on youtube from various eras, including the 80s, 90s, and current times. The playlist encompasses different genres and features songs from multiple artists.

To obtain the audio files of the songs in our playlist, the yt-dlp tool was utilized. We specifically downloaded the audio as .opus files, along with the corresponding thumbnail images and metadata in JSON format. The metadata includes information such as the song's ID, author, and URL.

To facilitate easy access to the song information, a separate JSON file was created that consolidates the extracted song details. This file serves as a reference for the resulting songs in our system.

4.2 Data preprocessing

The data preprocessing task began by decoding the .opus audio files into a format that could be processed. This involved using the FFmpeg tool to convert the files into .wave format and adjusting the sampling rate of the audio file to 8,000 Hz, while converting it to a mono channel to ensure compatibility with our system.

Subsequently, the audio was segmented into 1-second audio segments with a 50% overlap of 0.5 seconds between each segment. Our focus then shifted to the time-frequency representation of the audio to extract the features of each song at each time instance. To achieve this, each audio segment was converted into a log mel-spectrogram, which serves as the input feature of the audio segment for our neural network.

Creating the log mel-spectrogram began with applying a Short-Time Fourier Transform (STFT) to each segment, using a frame size of 1024 samples and a hop size of 512 samples. To reduce spectral leakage, a Hann window was applied for windowing. The magnitude of the complex-valued spectrum was then squared to obtain the power spectrum.

The power spectrum was then processed using Mel filter banks to extract the relevant feature bands that are perceivable by the human ear. Lastly, the output from the Mel filter bank was converted into a decibel scale, producing a log mel-spectrogram that captures the

crucial features of the audio.

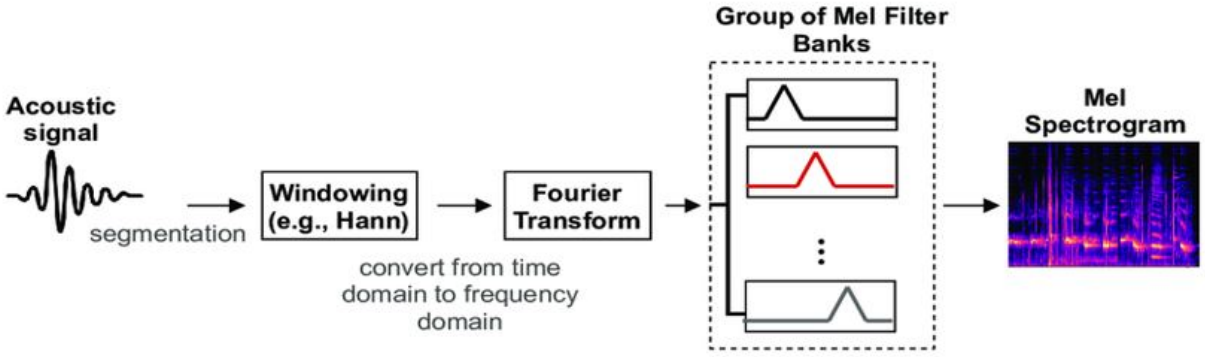


Figure 4.1: Overview of Audio preprocessing

4.3 Data augmentation

In our project, audio augmentation is done to generate the variations of the original songs so that both the original audio and its variant can be fed as input to train the neural network through contrastive learning. Although the audio augmentation process differs somewhat from that used for image data, the basic concept of generating variants remains the same. Some of the audio augmentation techniques used in our project are mentioned below:

- Background mixing: A randomly selected noise in the SNR range [0,10] dB is added to the actual audio to reflect the noise in audio during real-time query. the noise dataset as a subset of AudioSet[10] is used which includes the noise of the subway and metro.
- Reverb and microphone impulse response: To simulate the reverberation effect on the query audio and the impulse response of the recording equipment like a microphone, both room reverberation and microphone impulse response are sequentially applied by a convolution operation. For reverberation, the Aachen data set[11] is used and for microphone impulse response the MicIRP data set[12] is used.

4.4 Model architecture

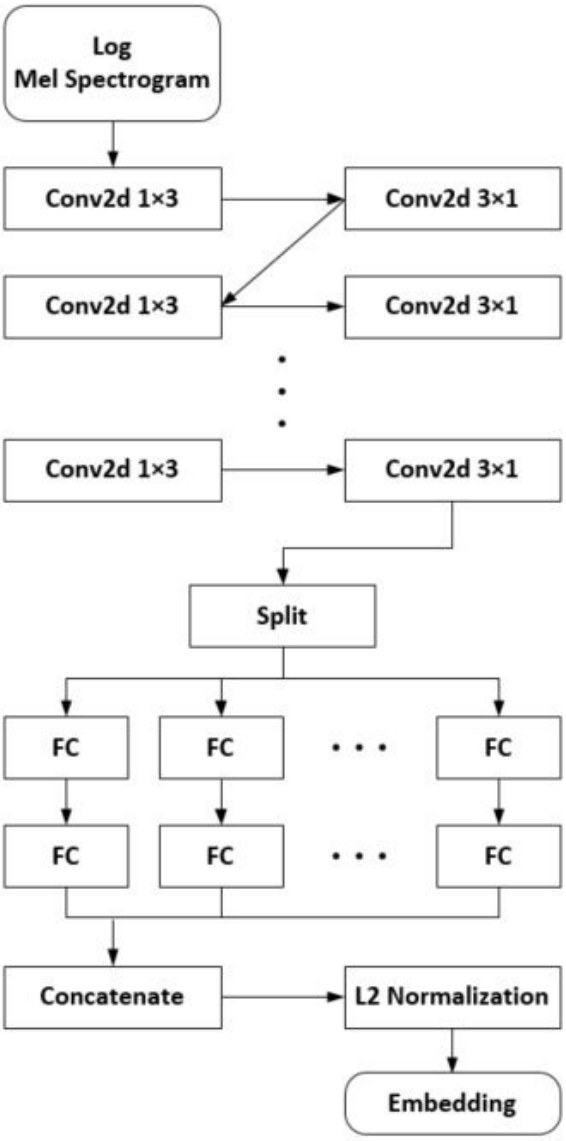


Figure 4.2: Model architecture

The audio fingerprint generating model as shown in figure 4.2 can be divided into two parts i.e. convolutional encoder and L2 projection layer. Input for this model is the log mel-spectrogram and the output of this model is an embedding vector(fingerprint) which is stored in the database to perform the nearest neighbor search with the query audio embeddings.

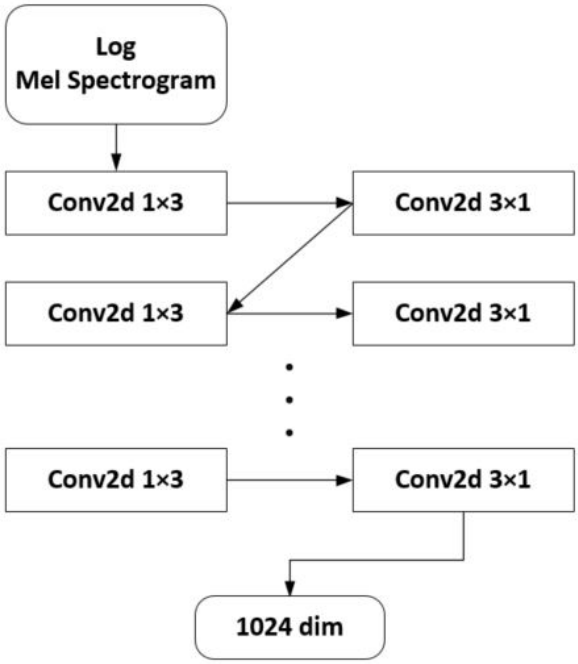


Figure 4.3: Encoder Layer architecture

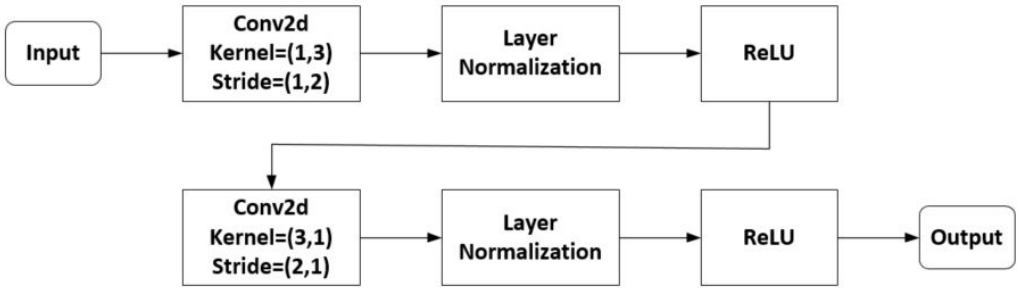


Figure 4.4: Spatially separable convolution layer

The convolutional encoder of this model as shown in figure 4.3 is just like that of the convolutional layer of CNN without the fully connected layer. It consists of multiple layers of spatially separable convolutions as shown in figure 4.4. Each layer of the spatially separable convolution consists of two convolutions. The first convolution is performed on the time axis of the spectrogram using the kernel of size 1×3 and stride of 1×2. The second convolution is performed on the frequency axis of the spectrogram using the kernel of size 3×1 and stride of 2×1. Both of these convolutions are followed by the Layer normalization[13] and ReLU activation function. Each layer of spatially separable convolution shrinks the image by 2 times and gradually increases the number of channels until the output of the last layer is 1×1 image with 1024 channels i.e vector of 1024-dimension.

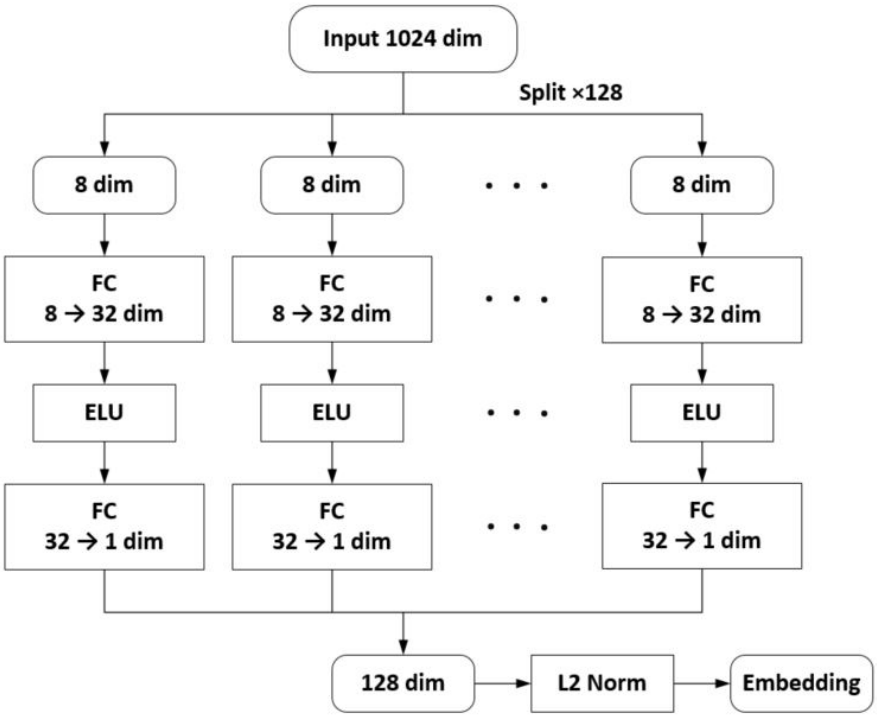


Figure 4.5: Projection layer

This 1024-dimensional vector is split into 128 8-dimensional sub-vectors, and these sub-vectors go through the different 2 layers(fully connected) of the neural network. This layer is called projection layer[5] or divide-and-encode layer[8] which is shown in figure 4.5. The first layer of the projection layer converts an 8-dimensional vector into a 32-dimensional vector followed by the ELU activation function. The second layer of the projection layer takes the 32-dimensional vector as the input and returns the 1-dimensional vector as the output. Finally, all the 1-dimensional output from 128 channels are concatenated to form a 128-dimensional single vector and this vector undergoes L2 normalization to get the final output as the embedding vector(fingerprint).

Also, the table 4.1 below shows the full detail of the model representing image size and channels out from each layer of the neural network.

Layer	Output size	Output channel
Input	256×32	1
Convolution layer 1	128×16	128
Convolution layer 2	64×8	128
Convolution layer 3	32×4	256
Convolution layer 4	16×2	256
Convolution layer 5	8×1	512
Convolution layer 6	4×1	512
Convolution layer 7	2×1	1024
Convolution layer 8	1×1	1024
Split into 128 sub-vector		8
L2 projection layer 1		32
L2 projection layer 2		1
Merge subvectors		128

Table 4.1: Model architecture

4.5 Training Algorithm

Configuration: All the hyperparameters like batch size(N), temperature($\mathcal{T}$), etc that are required for the audio fingerprint model are set. The values of all the hyperparameters are mentioned in the appendices of this report.

Input: For the audio fingerprint model, input is the log mel-spectrogram which is represented by 's'.

Output: Output is the embedding vector having dimension 'd which is represented by 'z'.

Augmenter:The augmenter is denoted by the $M_\alpha(.)$ which represents the chain of augmentation used in our project to create the augmented version of the original audio and α is the parameter set for the augmentation.

Network:This network model consists of two parts i.e encoder and L2 projection layer, each part is represented by the $f(.)$ and $g(.)$ respectively.

Steps involved in training the netwok:

1. Randomly take mini-batch of $N/2$ segments of the original song i.e. $\{s_k\}_{k=1}^{n/2}$
2. Perform data augmentation for each segment i.e.$M(s_k)$
3. Feed all the segments(original and augmented) to the neural network to get their corresponding embedding vectors z_k^{org} and z_k^{rep}.

$z_k^{org} = gof(s_k)$ where z_k^{org} is the embedding of the original segment

$z_k^{rep} = gof(M(s_k))$ where z_k^{rep} is the embedding of the augmented segment.

4. Take all the embeddings in pairs of original and augmented

   i.e. $z = \{(z_1^{org}, (z_1^{rep}, ..., (z_n^{org}, (z_n^{rep}\}$

5. Perform pairwise similarity between each of the embeddings in step 4 and calculate the loss for each pairwise similarity using the NTxent loss function.

$$l(i,j) = -\log \frac{\exp(a_i, j/\mathcal{T})}{\sum_{k=1}^{N} 1(k \neq i) \exp(a_i, j/\mathcal{T})} \tag{4.1}$$

6. Total loss L that averages a loss $l(i,j)$ of all positive pairs (pair of original and its augmented version) for both (i,j) and (j,i) is calculated

$$L = \frac{1}{N} \sum_{k=1} N[l(2k-1), l(2k, 2k-1)] \tag{4.2}$$

7. Parameters of $f(.)$ and $g(.)$ are updated to minimize loss L.

These whole steps are carried on entire training data.

4.6 Storing fingerprint and songs info in database

After training a neural network, an efficient model is obtained that generates audio fingerprints for songs. An audio preprocessing task is performed on each song to obtain log mel-spectrogram of each segment of the song. The log mel-spectrogram is then fed into the neural network, which generates embeddings for each audio segment.

These embeddings, which serve as fingerprints for the audio segments, are stored in a .mm file as a database. Additionally, a separate JSON file is used to store information such as the song title, author, and URL. These two files are linked together such that the song information and corresponding fingerprints can be easily accessed.

4.7 Searching and matching

In order to obtain the best match for an audio query, a search and matching process is initiated on the database of fingerprints. To enable this search, each fingerprint within the database must first be indexed. For this task, the FAISS[14] technology is used, and the IVFPQ method is chosen as it provides optimal results for the given constraints of storage, speed, and accuracy. The indexing process involves clustering all of the fingerprints in the

database, creating Voronoi cells. At search time, the n nearest Voronoi cell is determined by comparing the fingerprint of L consecutive segments of the audio query sequence $Q_{i=0}^{L}$ with the centroid of the Voronoi cell. Subsequently, the actual search is performed on all the fingerprints lying within the n nearest Voronoi cell. Product Quantization[15], which involves reducing the dimension of the vector, is applied to the fingerprint, resulting in efficient storage and speed, although it may slightly impact accuracy. In this process, each fingerprint is divided into sub-fingerprints, and clustering is performed, creating b groups. Each sub-fingerprint is then replaced by the nearest centroid of the sub-fingerprint cluster, resulting in each sub-vector being quantized into b bits.

The search process results in obtaining the top k segment level search result indices I_{Q_i} of fingerprints for each 1s segment query Q_i. The offset is then compensated by $I'_{Q_i} = I_{Q_i} - i$. The set of candidate indices $c \in C$ is determined by taking unique elements of I'_{Q_i}. The sequence-level similarity score is obtained by summing all segment-level similarities from the segment index range $[c, c + L]$, and the index with the highest score is determined as the best-matched result.

4.8 System architecture overview

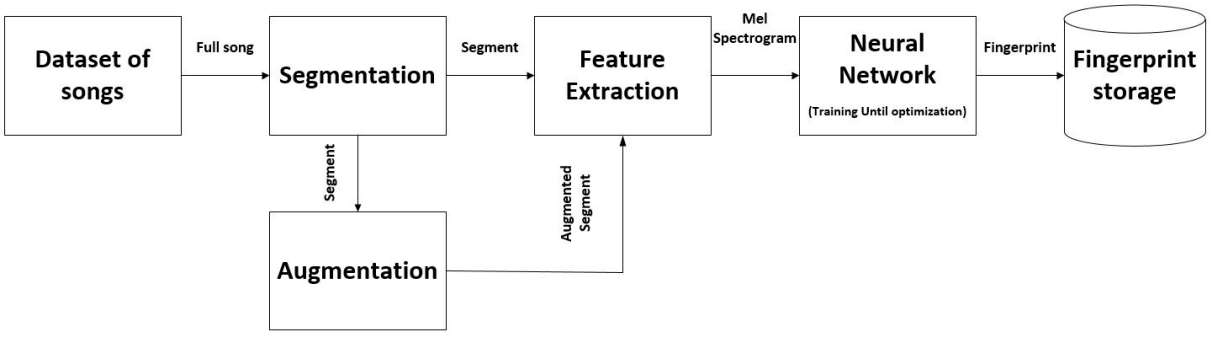


Figure 4.6: Fingerprint generation and storage architecture

The architecture depicted in figure 4.6 represents a standard approach for generating and storing audio fingerprints in a music recognition system. The process involves preparing a dataset of Nepalese songs and segmenting the original songs into smaller segments. The segments are then augmented to increase the variability of the data.Next, the feature extraction block converts the audio segments(both original and augmented) into log mel-spectrograms, which represent the frequency and intensity of the sound waves over time. These spectrograms are then fed into a neural network block, which is trained using a contrastive learning approach to generate optimal embeddings for each segment. After training neural networks

for the number of epochs, the optimal network is obtained and used to generate the embeddings(fingerprint) of each audio segment. These fingerprints are stored in a .mm file, which serves as the database for the music recognition system.

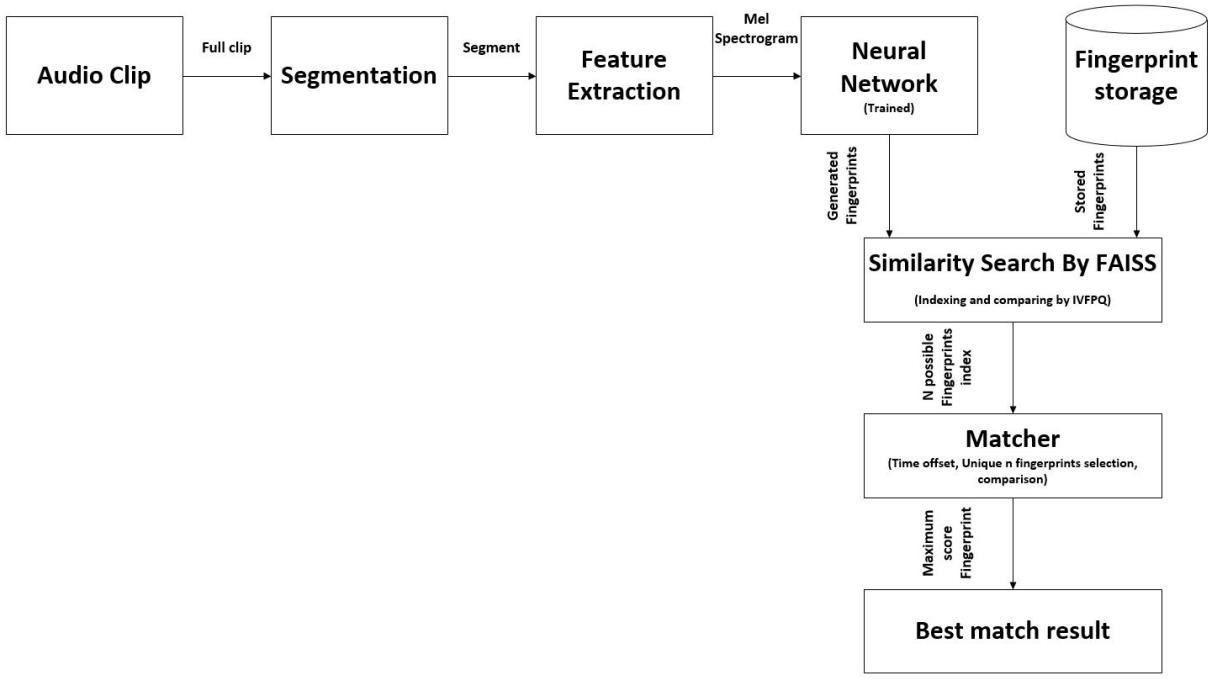


Figure 4.7: Query searching and matching architecture

The architecture depicted in figure 4.7 outlines the process for searching and matching recorded input audio (query) in a database of stored fingerprints. The process begins with the user accessing the microphone in their mobile device and using a mobile app to send a clip of a song as the query. Once the query clip is received, it is sent to the segmentation block, which partitions it into smaller sections for processing. The feature extraction block then converts each segment into a log mel-spectrogram representation, which is used by a well-trained neural network to generate fingerprints for each segment. The recently generated fingerprints are indexed and searched using FAISS technology, which returns a number of best searches from the database. These search results are sent to the matcher block, which performs a time offset of each input fingerprint(search result) and identifies unique fingerprints as candidate fingerprints. Ultimately, the matcher block calculates the cosine similarity between each candidate fingerprint and the fingerprints of the query clip. The candidate fingerprint with the highest similarity score is considered the final result.

4.9 Use Case diagram

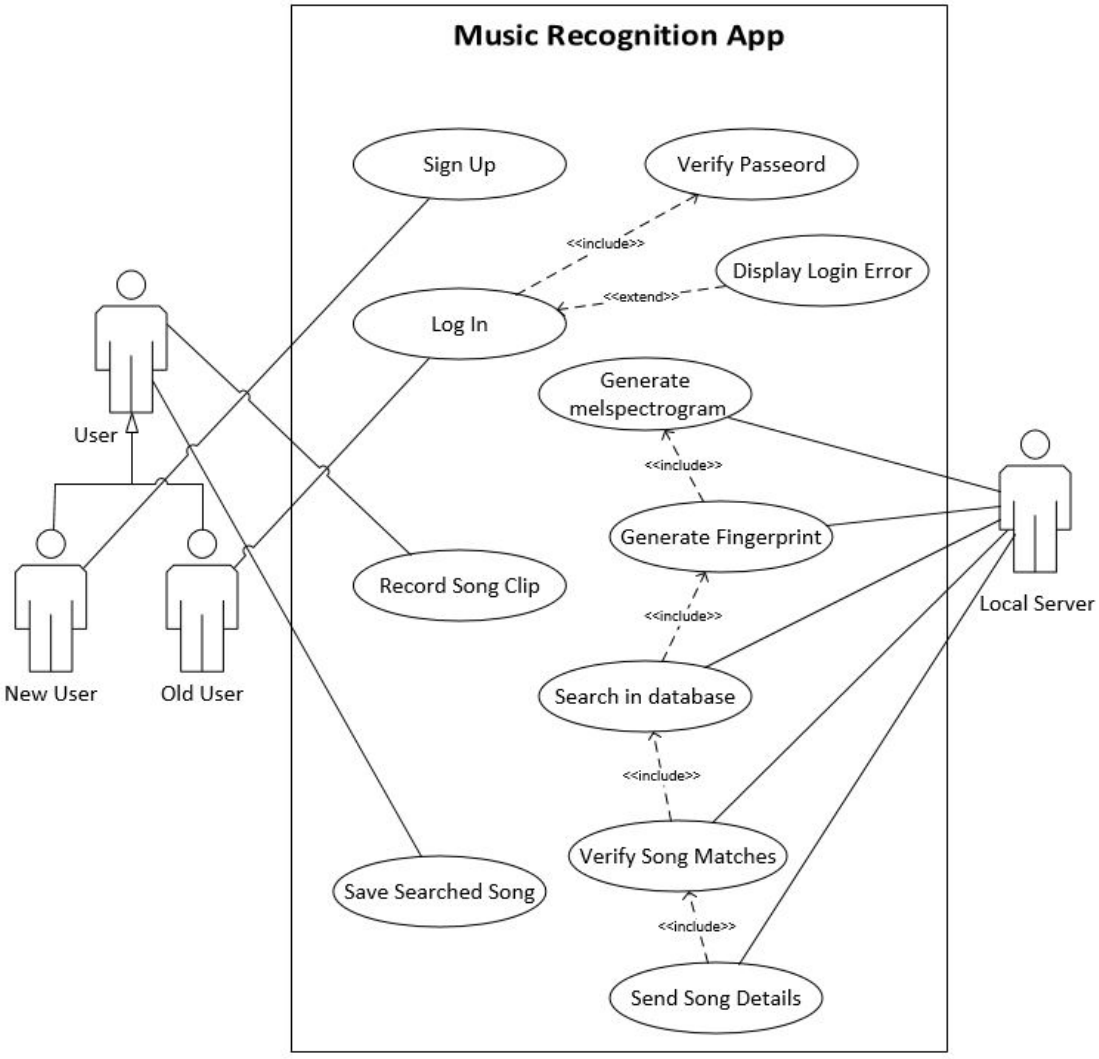


Figure 4.8: Use Case diagram

The diagram in Figure 4.8 illustrates the use case scenarios for the music recognition system, which includes actors, use cases, and their relationships. If the user is new to the system, they must sign up to gain access. Otherwise, the user must log in by providing their correct credentials. After logging in, the user has the option to record a song clip or save a previously searched song. On the local server, tasks such as log mel-spectrogram generation, fingerprint generation, searching, and matching are performed. Once the best match is verified, details of the song are sent to the user.

4.10 Activity diagram

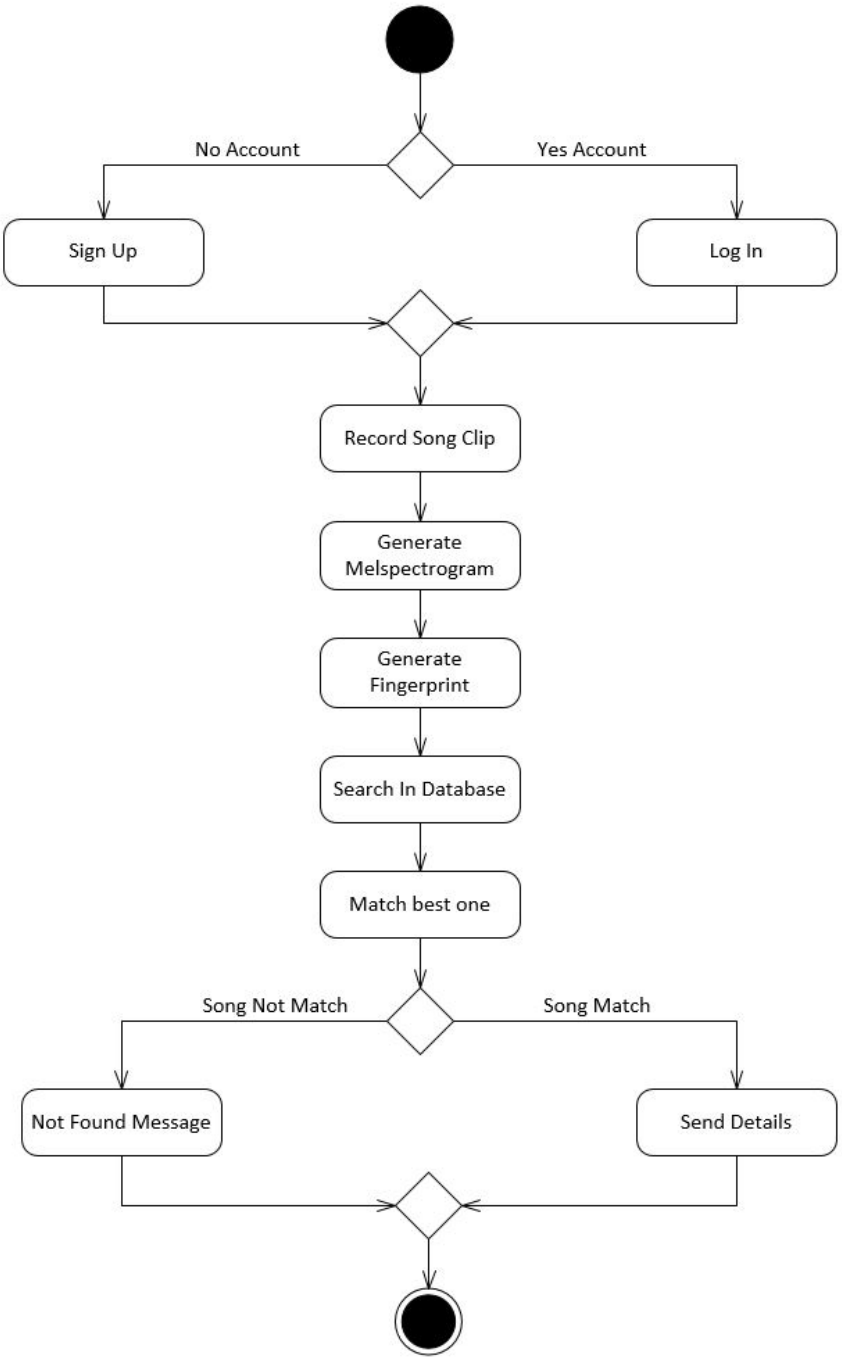


Figure 4.9: Activity Diagram

The diagram in Figure 4.9 is an activity diagram that illustrates the flow of actions in the music recognition system. To begin, the user can either sign up for an account or log in to an existing account. Next, the system records the song clip and generates a mel-spectrogram

and fingerprint for the clip. The system searches the database for the best match and returns details about the song if it is found. If the song is not found in the database, the system returns a message indicating that it could not be found.

4.11 Sequence diagram

The diagram in Figure 4.10 depicts a sequence diagram that represents the sequence of events in the music recognition system. The diagram shows the messages being passed from one object to another and the activation period of each object in the system. If the user is not registered, the system presents a sign-up page in the user interface (UI) to the user. The user can provide their details, which are then recorded in their profile. If the user is already registered, they can input their credentials to log into the system. If the credentials are correct, the user can enter the system. Once the user is inside the system, they can access the microphone of their mobile device by tapping a button on the UI and record a song clip. The system performs all the necessary tasks, such as log mel-spectrogram generation, fingerprint generation, searching, and matching, in sequential order. If the system finds a match for the recorded song, it retrieves and displays the information about the matched song in the UI. The user can also save the searched song in their profile for future reference.

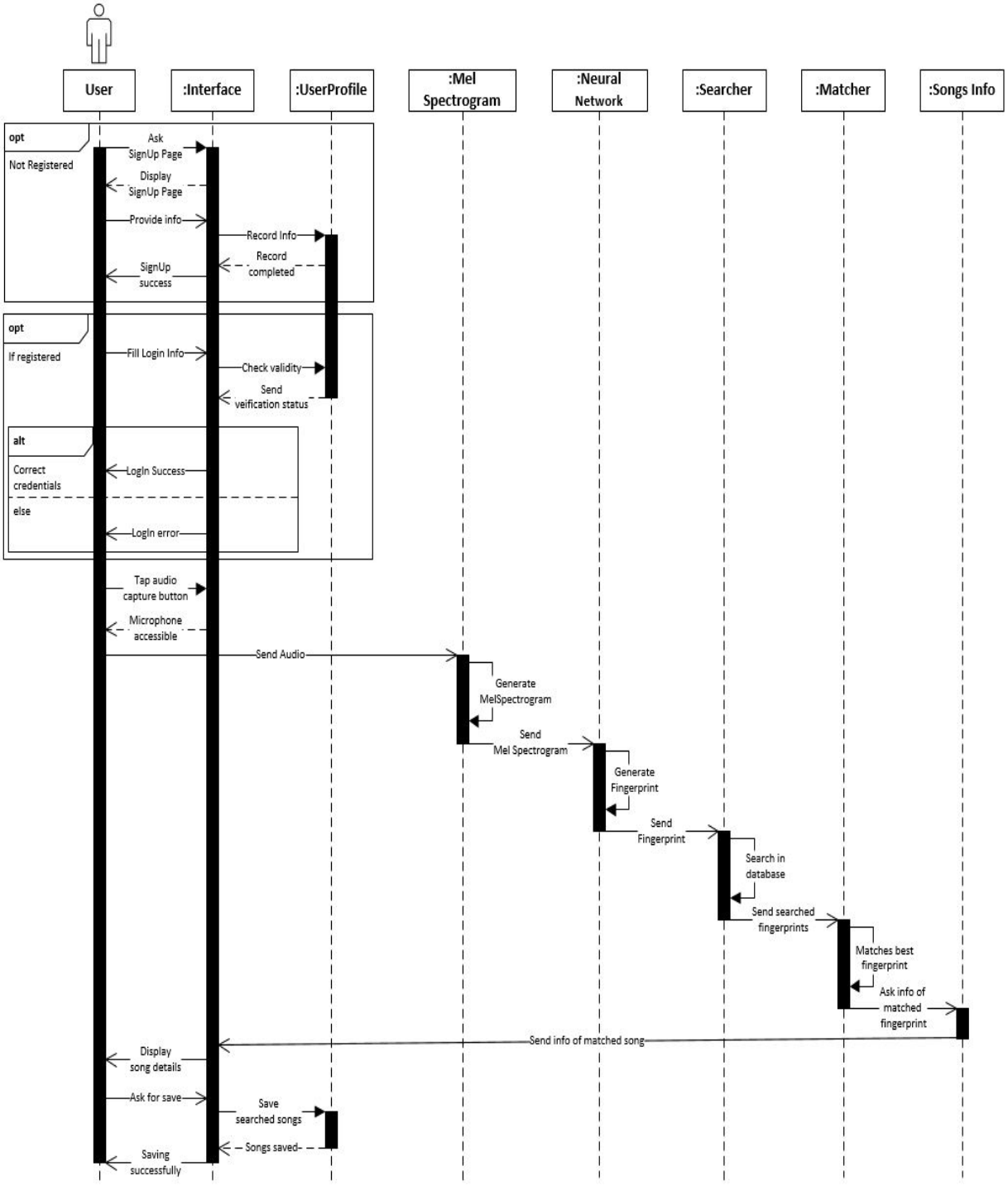


Figure 4.10: Sequence Diagram

4.12 Technologies Used

4.12.1 Python

In our project, we used python programming language to build the system. It is a versatile and powerful programming language that has a large and active community of developers who contribute to its development, documentation, and support. Such a community also somehow helped in our project. The main reason for choosing python for our project is due to its key features like readability, simplicity, and consists large standard library that provides developers with a wide range of tools and functions, as well as a large and growing number of third-party libraries and frameworks.

4.12.2 TensorFlow

TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks, including machine learning, deep learning, and neural networks. It enables researchers and developers to build and train machine learning models quickly and easily. We used this library in our project to build and train our neural network model consisting of various layers.

4.12.3 Kapre

Kapre is an open-source software library for audio processing with deep neural networks. We have used this library in our project for audio processing and generating the log mel-spectrogram (audio feature) from the raw audio. Various inbuilt functions like STFT, Magnitude, ApplyFilterBank, etc were imported from this library in our project.

4.12.4 Flutter

In our project, we used flutter for mobile app development which is an open-source mobile application development framework that provides a powerful and efficient way to create cross-platform mobile applications that can run on both Android and iOS platforms using a single codebase. It is based on the dart programming language. Its features like cross-platform, fast development, high performance, beautiful UI and active community attracted us to use flutter in our project.

4.12.5 Google Colab

Google colab is a cloud-based platform for developing and running Python code in a web browser. In our project, we built and train our model in Google colab. Its pre-configured environment with popular data science libraries and access to powerful hardware like GPUs

and TPUs made it easy to build our model and accelerate the training of the model. Its best thing is that we can use it free of cost which is worth completing a project within a small budget.

4.12.6 Microsoft Visio

In our project, we used Microsoft Visio to create all the diagrammatic stuff needed in our project. Using the templates and various shapes provided by Microsoft Visio we draw system architecture, use case diagram, activity diagram, and sequence diagram effectively.

4.12.7 yt-dlp and ffmpeg

Yt-dlp is an open-source command-line program/tool which we have used in our project to download audio, thumbnail and metadata of music videos on youtube. FFmpeg is a cross-platform open-source software that is used for processing and converting audio and video files. We used it in our project to convert .opus audio format into .wav format and also changed the sampling rate to 8000Hz with a mono channel.

4.12.8 Visual Studio Code

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft. Its features like lightweight, fast, highly customizable, and availability of extensions/plugins have really made us easy for coding.

4.12.9 FAISS

FAISS (Facebook AI Similarity Search) is an open-source library developed by Facebook's AI Research team for efficient similarity search and clustering of high-dimensional vectors. It is used in our project because of its simplicity and its ability to perform fast and efficient nearest neighbor searches in large datasets containing millions or billions of high-dimensional vectors.

5. Epilogue

5.1 Discussion and Result Analysis

5.1.1 Training Data

We utilized two datasets to train our model. The first dataset, called FMA, consisted of 10,000 English songs that were 30 seconds in length. We used 500 of these songs for querying and database searching in order to evaluate the performance of the model. The second dataset contained 100,000 English songs that were also 30 seconds in length. We used this dataset to scale the database for the evaluation process.

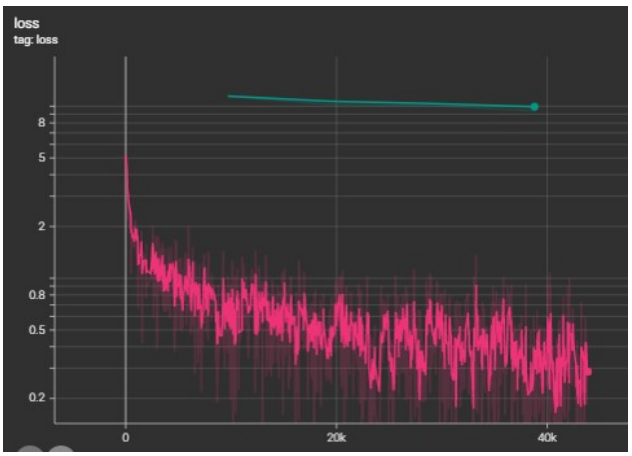


Figure 5.1: Train data using FMA dataset

Training the model using FMA dataset gave us the losses as shown in the fig 5.1. The results of our training and validation loss were similar to those reported in the neural audio fingerprinting research paper. Specifically, we observed that the training loss was low, but the validation loss remained high.

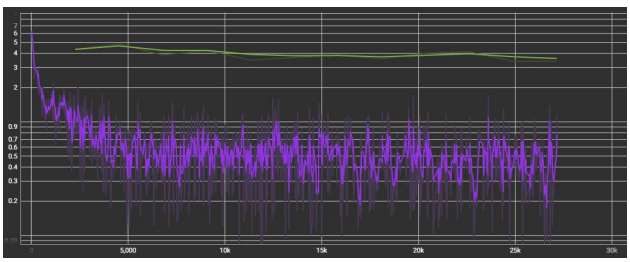


Figure 5.2: Train data using Nepali dataset

We subsequently developed a new model by training it on a Nepali dataset. This involved collecting Nepali songs from YouTube and providing them to the model. Initially, we had a dataset of 200 Nepali songs, with an average length of 4 minutes. We later expanded this dataset to almost 900 songs, also of 4 minutes length. Our results indicated that the training and validation loss were similar to those obtained from training the model on the FMA dataset. However, we observed that the validation loss was higher than the training loss as shown in fig 5.2

5.1.2 Evaluation

To measure the performance in segment/songlevel search, we use Top-1 hit rate(

$$\frac{\sum_{i=1}^{N} I(p_i \in T_i)}{N} \times 100$$

where N is the total number of predictions, p_i is the i-th predicted item, T_i is the set of ground truth items for the i-th query, and I is the indicator function that returns 1 if the predicted item is in the set of ground truth items, and 0 otherwise.

Segment (Seconds)	**1(1s)**	**3(2s)**	**9(5s)**	**11(6s)**
Top1 exact	61.73	84.15	95.60	96.59
Top1 near	64.04	84.86	95.66	96.61
Top3 exact	68.65	87.58	96.56	97.39
Top10 exact	73.29	89.29	97.09	97.84

Table 5.1: Top1 hit rate (%) of segment-level search

For test-query generation, we randomly selected 2,000 query-sources for each clip length of 1, 2, 5 and 6 seconds. These query-sources were cropped from a Test-DB that contained 500 clips, each with a duration of 30 seconds. To synthesize each query, we utilized a random augmentation pipeline, where the default SNR range was set to [0, 10] dB. We ensured that the data used for background mixing and IR were unseen by our model by isolating them from the training set. The longer the query sequence, the better the performance was given by the model.

5.1.3 Mobile App

5.1.3.1 Features of mobile app

1. User Authentication: This system allows users to either sign up for a new account or access an existing one by providing their email address or social media login credentials. Additionally, users have the option to reset their password in the event that they forget it.

2. Audio Record: By using the application's audio recording feature, users can search for details about a song. They can record the audio within the app and use it to identify the song through a search function.

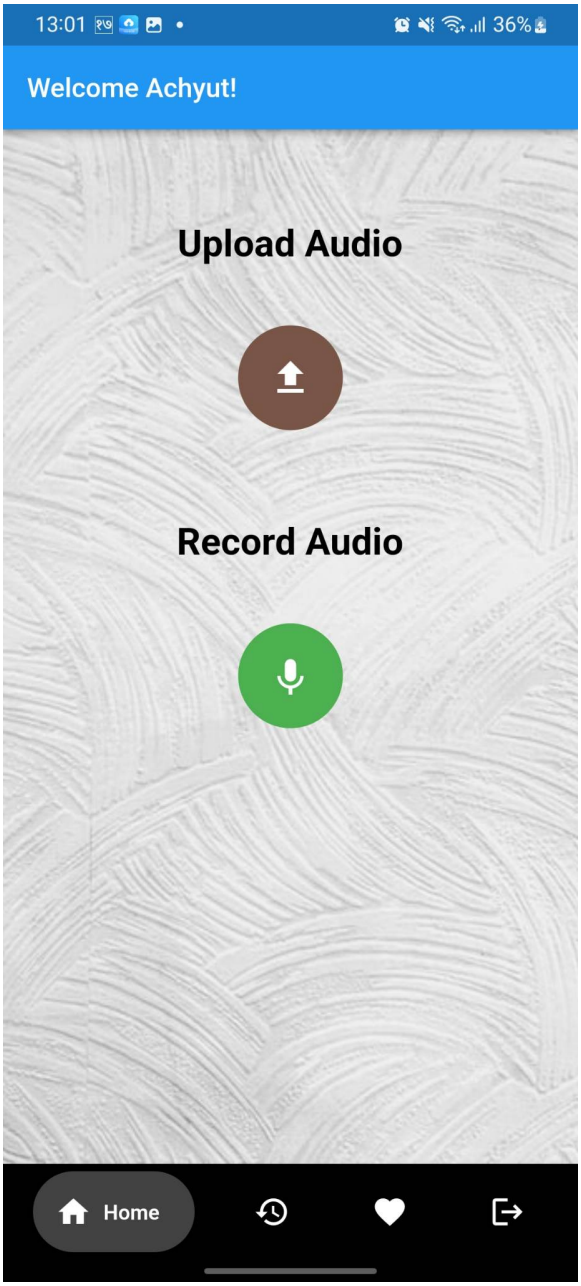
3. Audio Upload: The application enables users to upload audio files from their device, which can be utilized for searching or saved onto their profile for future reference.

4. History: Upon searching for a song, the application stores the search results in the user's history, which can be accessed and searched at any time by the user.

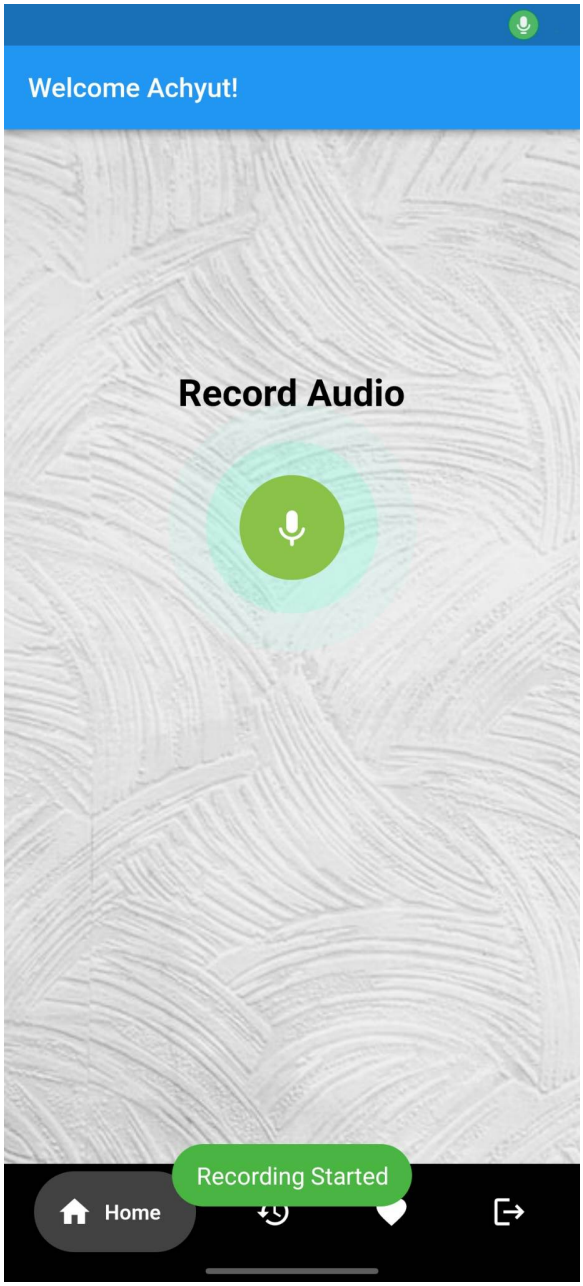
5. Songs identification: The primary function of the mobile application is to identify songs based on audio clips submitted by users.

6. Interactive interface: The interface of the application has been designed to be user-friendly, allowing for easy navigation and comprehension by all users

7. Recommend songs: In addition, the application suggests songs of a similar genre to the search result as well as other songs by the same artist.

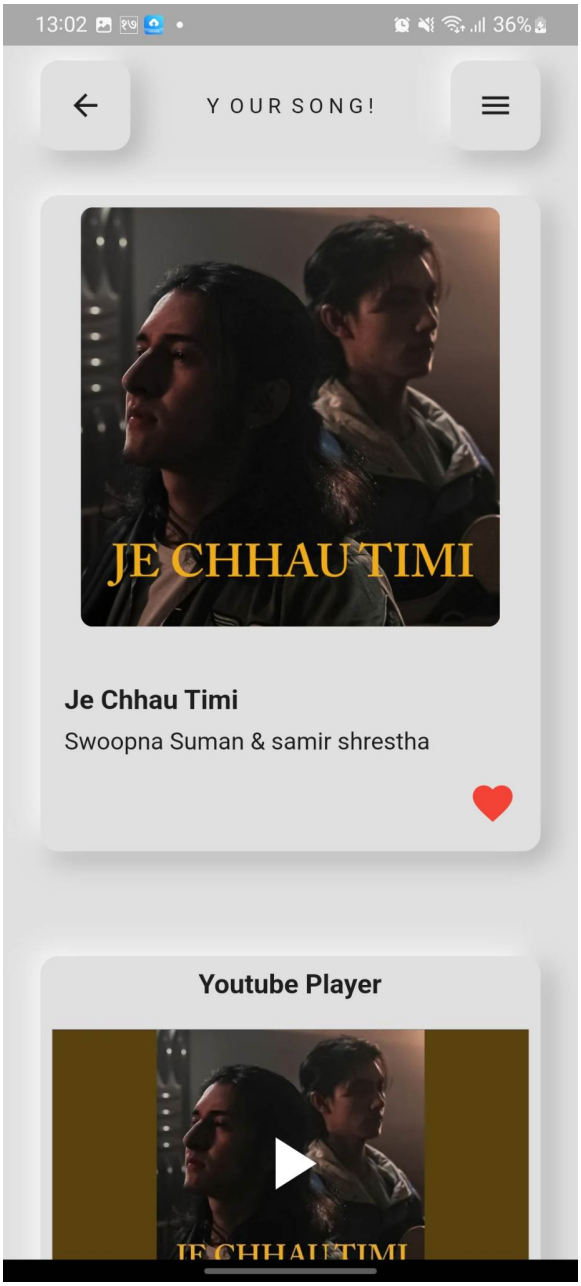


(a) Home Page

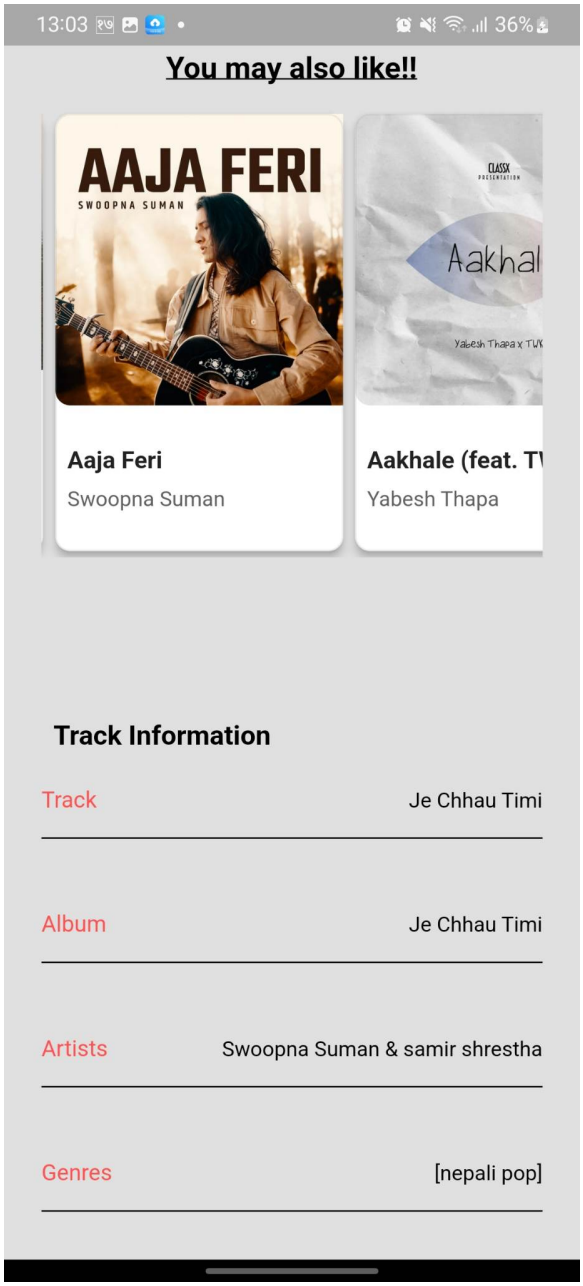


(b) Recording

Figure 5.3: Home Page and Recording Page

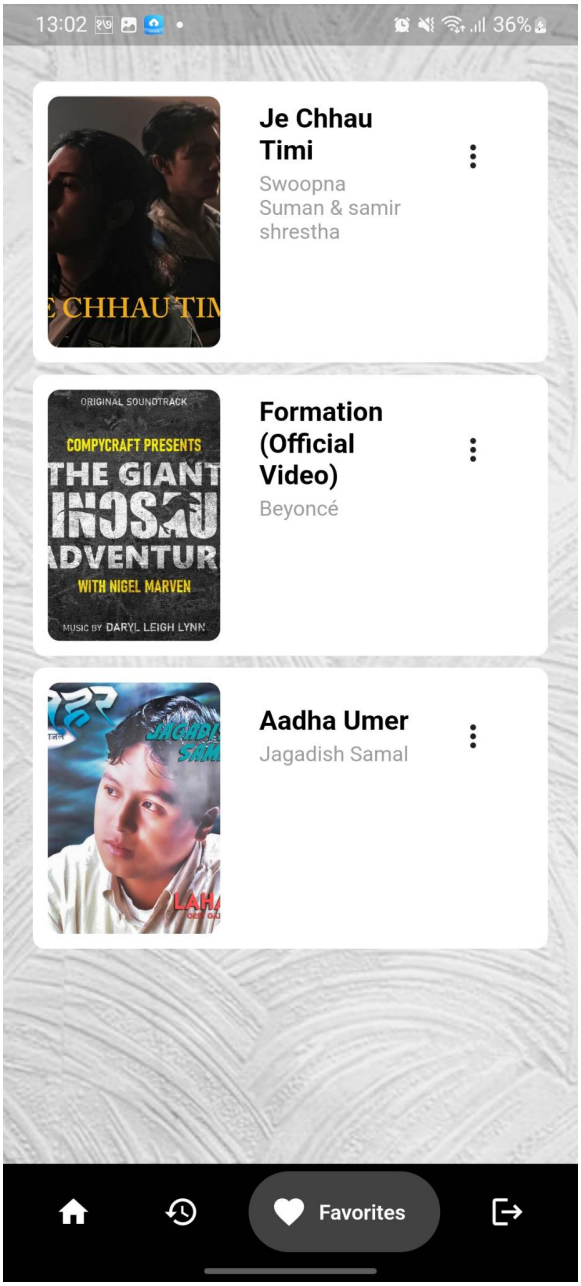


(a) Result Screen 1

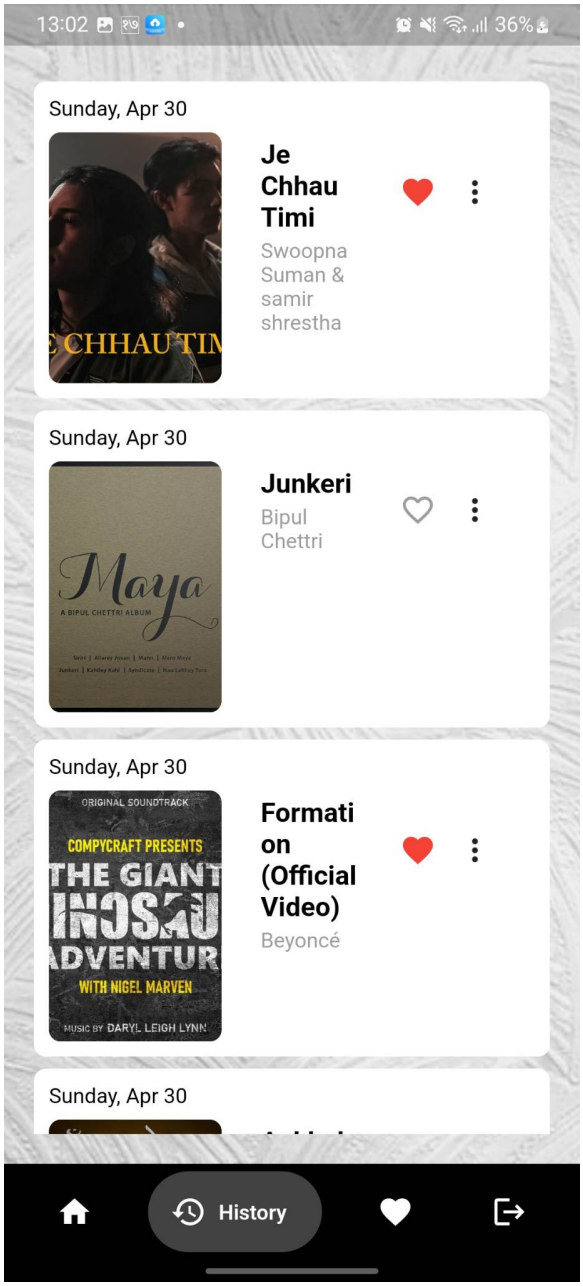


(b) Result screen 2

Figure 5.4: Result Screens



(a) Favourites Page



(b) History Page

Figure 5.5: Favourites and History Page

5.2 Conclusion

Our study involved the development of a music recognition system integrated into a mobile app, which enables users to record songs of any duration and retrieve information such as the song's title, author, and URL. This system is built on a deep neural network using a contrastive learning approach to train the network. Specifically, log mel spectrograms of each segment of an original song and its augmented version generated using various signal processing techniques were used as training data. Audio fingerprints(embeddings) were generated using the trained model and stored in a database that can be searched to match the audio query fingerprint. We employed the FAISS library for similarity search and indexing of the stored fingerprints in a large database.

We applied various augmentation techniques, such as adding noise and impulse response, to simulate real-time audio query conditions. Our results showed that the system is capable of accurately recognizing music from diverse sources, but we acknowledge that there is still room for improvement in terms of performance.

5.2.1 Limitations

Despite the best efforts of our team, the music recognition system has several limitations that we recognize and aim to address in future research. One of the major challenges is maintaining the system's accuracy, responsiveness, and scalability. As the system grows, it will be important to ensure that it can handle a large number of queries and continue to provide reliable results.

Another limitation of the current system is the relatively small number of songs in the database, which restricts its ability to recognize a broader range of music. It is acknowledged that increasing the size of the database will require significant resources and careful consideration of ethical and legal issues surrounding music copyright.

Furthermore, it is recognized that the system's performance may be limited by the fact that it has not been trained on all genres of music. Complex structures found in songs of certain genres may make them more difficult to accurately recognize. Future research will address this limitation by exploring ways to incorporate a wider variety of musical styles in the training data, as well as exploring the use of transfer learning[16] techniques to improve the system's ability to recognize new genres.

5.2.2 Future enhancements

Although the music recognition system developed in our project is capable of recognizing songs and providing expected outcomes, there is still room for further advancements through various enhancements.

The following future enhancements have been identified for our project:

- Include a large number of songs in the database
- Automatic training of model for new songs.
- Make the system able to recognize the sung/hummed song too.
- Making comparative studies on various models to find the optimal model for music recognition.

Overall, these enhancements have the potential to improve the accuracy, efficiency, and usefulness of music recognition systems for both consumers and businesses.

References

[1] Shazam. https://www.shazam.com/.

[2] SoundHound Inc. Soundhound. https://www.soundhound.com/.

[3] Google assistant. https://assistant.google.com/services/a/uid/00000024216d4bb8?hl=en-US.

[4] Jaap Haitsma and Ton Kalker. A highly robust audio fingerprinting system. In *International Society for Music Information Retrieval Conference*, 2002.

[5] Sungkyun Chang, Donmoon Lee, Jeongsoo Park, Hyungui Lim, Kyogu Lee, Karam Ko, and Yoonchang Han. Neural audio fingerprint for high-specific audio retrieval based on contrastive learning, 2020.

[6] Mandla VamshiKrishna and Dasika Moukthika. Study on framework of audio fingerprinting and shazam's working algorithm. *International Journal of Advanced Research*, 6:690–702, 11 2018.

[7] Avery Wang. An industrial strength audio search algorithm. 01 2003.

[8] Blaise Agüera y Arcas, Beat Gfeller, Ruiqi Guo, Kevin Kilgour, Sanjiv Kumar, James Lyon, Julian Odell, Marvin Ritter, Dominik Roblek, Matthew Sharifi, and Mihajlo Velimirović. Now playing: Continuous low-power music recognition, 2017.

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[10] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[11] Marco Jeub, Magnus Schäfer, and Peter Vary. A binaural room impulse response database for the evaluation of dereverberation algorithms. pages 1 – 5, 08 2009.

[12] Microphone impulse response project. http://micirp.blogspot.com/.

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[14] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[15] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[16] Mohammadreza Iman, Khaled Rasheed, and Hamid R. Arabnia. A review of deep transfer learning and recent advancements, 2022.

Appendices

Parameters	Values
audio format	.wav
sampling rate	8,000Hz
channel	mono

Table 5.2: Parameters of audio properties

Parameters	Values
segment duration	1s
overlap duration	0.5s

Table 5.3: Parameters of segmentation

Parameters	Values
STFT window size	1024
STFT hop size	256
STFT window function	Hann
Mel frequency range	300-4000Hz
log mel spectrogram size(F×T)	256×32

Table 5.4: Parameters of log mel spectrogram

Parameters	Values
Batch size(N)	120
learning rate	10^{-4}
optimizer	Adam
epochs	15
learning rate schedular	cosine decay to 10^{-7}no warmup & restart
fingerprint dimension(d)	128
Convolution encoder output(h)	1024
splitting number	128

Table 5.5: Hyperparameters of neural network model

Parameters	Values
Voronoi cells number	256
quantization bit	8
number of best search result	20

Table 5.6: Parameters for searching