International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

Deep Learning-based Fingerprinting Methods for Audio Representation and Search

Divesh Singh

Infosys Limited, Mumbai, India

***Abstract* - Audio content is abundant and diverse in today's digital age, ranging from music to podcasts and audio streams. Efficiently representing and searching this vast audio data is essential for applications like content identification, recommendation systems, and audio retrieval. Traditional audio fingerprinting methods have relied on handcrafted features and heuristics, which may lack scalability and robustness in real-world scenarios.**

In contrast, deep learning has shown remarkable capabilities in various audio-related tasks, such as speech recognition and music classification. Leveraging deep learning-based methods for audio fingerprinting offers the potential to create compact yet informative representations of audio signals, enabling faster and more accurate content identification and search.

This paper explores deep-learning model to develop advanced audio fingerprinting methods. By utilizing models such as a variant of autoencoders – U-Net Autoencoders and Convolutional Neural Networks (CNNs), the work in the paper seeks to extract audio features, and compress and encode them to reduce the feature space effectively. Also, the work scope includes the challenge of noise resilience, ensuring that the audio fingerprints remain consistent and robust even for noisy samples.

This compressed, encoded audio fingerprint is then used to efficiently search the audio database for required purposes (for example, music identification). For creating the audio database, vector database of FAISS is selected as it provides efficient vector search capabilities, which can be utilized well for music identification.

Keywords: Efficient representation of audio, audio fingerprinting, deep learning, U-Net Autoencoders, Convolutional Neural Networks (CNNs), compact feature representation, noise resilience, audio analysis, audio database search, vector database, Facebook AI Similarity Search (FAISS).

I. INTRODUCTION

1.1 Background and Context

The current digital era has witnessed an unprecedented explosion in the volume and accessibility of audio content. From music streaming platforms to voice assistants, audio plays a ubiquitous role in our daily lives. However, with this abundance of audio data comes the challenge of efficient audio content identification and retrieval [1]. When it comes to managing and searching audio content, the conventional approaches often struggle to keep up with the ever-expanding audio data [2][3].

In this context, the research and methodology presented in this paper delves into the deep learning based fingerprinting model for audio representation and search. It addresses the critical need for compact representations of audio signals to facilitate rapid and accurate content identification. Through the lens of deep learning, specifically the Autoencoders, this paper explores innovative techniques to create resilient audio fingerprints that can withstand the challenges posed by noise and variability in audio recordings.

Further, it explores methodologies of storing fingerprints of the songs in database, so that song identification can be performed against this database.

1.2 Scope of Work

The work scope includes exploring and implementing deep learning techniques to develop advanced audio fingerprinting method. By utilizing approaches such as autoencoders and CNNs, the work seeks to extract audio features, and compress and encode them to effectively reduce the feature space.

Also, the work scope includes the challenge of noise resilience, ensuring that the audio fingerprints remain consistent and robust even in presence of noise.

This encoded data can then be used to search the audio database for required purposes (for example, music identification). This involves selecting a suitable database type for creating audio database using suitable approaches so that fingerprints can be used to query this data.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

II. LITERATURE REVIEW

This chapter surveys the academic literature in the field of audio fingerprinting and content-based audio retrieval, laying the foundation for the deep learning-based audio fingerprinting methods explored in this paper. By examining existing research, methodologies, and challenges, this section highlights the progression of audio content identification techniques and sets the stage for the innovative approaches discussed in later chapters.

2.1 Audio Fingerprinting and Content-Based Retrieval

Audio fingerprinting is a pivotal technique in audio content identification and retrieval. Traditionally, audio fingerprinting systems have relied on handcrafted features and manual annotation. One such system, proposed by Avery, Li-Chun, and Wang in 2004 [1], introduced an industrial-strength audio search algorithm. This algorithm aimed to efficiently match audio snippets to a reference database, marking an early step toward scalable audio content identification. This paper outlines detailed approach to finding features using spectrogram and combinatorial approach for storing and searching features which forms the basis of Shazam application.

Haitsma and Kalker's work in 2002 [2] further contributed to the field by introducing a highly robust audio fingerprinting system, wherein the authors discuss on the mathematical models such as Hidden Markov Model and other quantization models for constructing fingerprints. And proposed an approach that uses sub-fingerprints generated from the global energy of each interval and the spectral decomposition of each frame into logarithmically spaced bands. The Hamming distance between the sub-fingerprint sequences of two audio samples is used to measure their similarity. However, noise and alterations can corrupt the sub-fingerprints, which in turn corrupts the Hamming distances and reduces the accuracy of indexing algorithms. Possible solutions proposed are using perpetual filters to denoise or using a robust fingerprint extractor. This work focuses on the designing of this robust fingerprint extractor.

2.2 Deep Learning in Audio Fingerprinting

Deep learning has rapidly gained prominence in recent years as a transformative force in audio fingerprinting. Panyapanuwat, Kamonsantiroj, and Pipanmaekaporn (2021) [3] presented an approach to content-based audio retrieval using unsupervised deep neural networks. The authors evaluated the approach on two tasks: music identification and music retrieval by using fingerprint hashing functions along with Deep Neural Network for compression. The proposed method created a compact representation of the spectrogram into a 32-bit vectors.

Choi, K., Fazekas, G., Cho, K. and Sandler, M. (2017) [4] discussed theoretically on how deep learning can work for working for audio data to various use cases such as Music tagging and Genre classification. The Appendix in the paper touches upon new models which can be used for working with audio data

2.3 Data set for working with noisy audio

Valentini-Botinhao, Cassia (2017) [5] offers a clean and noisy audio database set, specifically designed for training and evaluating speech enhancement algorithms operating at 48 kHz. Its versatility extends to other audio data processing applications.

2.4 FAISS vector database

Johnson, J., Douze, M., & Jégou, H. (2017) [6] provides details on a library for efficient similarity search and clustering of dense vectors. It discusses the basic concepts of similarity search and clustering. It also details the various algorithms implemented in the Facebook AI Similarity Search (FAISS) library. It acts as a vector database which can be used to create database of audio fingerprints to be queried upon.

2.5 Summary

In summary, the literature review underscores the evolution of audio fingerprinting methods, from traditional techniques to the promising deep learning-based approaches. While early algorithms laid the groundwork for audio content identification, the latest developments in deep learning have opened new horizons for efficient, noise-resilient audio fingerprinting. Also, the suitable vector database is identified which can be used for creating audio database.

The subsequent chapters of this paper build upon this rich body of research, proposing innovative deep learning-based audio fingerprinting to further improve audio content representation and search, offering efficient, noise-resilient solutions for contemporary audio applications.

Based on above discussions in research papers on deep learning methods, this paper focuses on a modification of general autoencoders – U-Net Autoencoders [7], which requires converting audio data using Mel-Spectrogram transformation for a reliable audio fingerprint generation. This autoencoder architecture also helps in denoising image samples [8] (which can be extended to audios for our use case). The details are in following chapters.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

III. PROPOSED METHODOLOGY

3.1 Data Collection and Preprocessing

The primary dataset utilized in this work/project is the "Noisy speech database for training speech enhancement algorithms and TTS models" [5], comprising clean and noisy audio tracks.

3.2 Data Preprocessing

Before feeding the audio data into deep learning models, extensive preprocessing is performed for both, clean and noisy audio samples. This involves:

- Resampling audio clips to ensure a consistent sample rate (e.g., 48 kHz)
- If the audio signal has more than one channel, it takes the mean of the channels
- If the audio signal is longer than the desired duration, it crops it to the desired duration
- If the audio signal is shorter than the desired duration, it pads it with zeros
- Feature extraction to convert raw audio waveforms into informative representations - Mel-spectrogram (normalized and denormalized) transformation is used in this work. This also aids in normalizing audio signals if to_normalize_mel_spectrogram configuration is enabled
- The configurations used for preprocessing the data are:
  - to_normalize_mel_spectrogram - True/ False – This affects number of epochs for training the model
  - batch_size=128 - Batched the data for faster training, each batch containing 128 audio samples
  - target_sample_rate=48000 - Adjusts the audio sample to this sample rate
  - n_fft=1024 - determines the number of data points in each short-time window of the spectrogram
  - duration=4 - Each audio sample is clipped or extended to 4 seconds
  - hop_length=512 - determines the step size between successive windows in the spectrogram calculation
  - n_mels=64 - specifies the number of Mel filter banks to be used when computing the Mel spectrogram. Mel filter banks are used to transform the linear-scale frequency data obtained from the FFT into a logarithmic Mel scale, which better approximates human auditory perception of sound [9]
- Example scenario for how an audio sample is transformed when these values are passed in Mel-Spectrogram transformation
  - Number of samples for a 4 second audio at 48 kHz sample rate = 4 * 48000 = 192000
  - Number of frames = (Number of samples - n_fft) / hop_length + 1 = (192000 – 1024) / 512 +1 = 374 frames
  - Number of mel frequencies = 64
  - Final shape of the output Mel spectrogram tensor = (n_mels, number of frames), which is (64, 374).

This tensor is used as input to the model

3.3 Deep Learning Models

One of the core requirements of this work involves the utilization of deep learning architectures, specifically autoencoders and convolutional neural networks (CNNs), to create robust audio fingerprints.

3.3.1 Autoencoder

Autoencoders are a class of neural network architectures designed for unsupervised learning tasks, where the primary goal is to learn compact representations of input data. They consist of two main components: an encoder, responsible for mapping input data to a lower-dimensional latent space (encoded signal or hidden representation); and a decoder, which aims to reconstruct the input data from the latent space [4].

The encoded signal is our point of interest for this work as this constitutes the fingerprint of the audio.

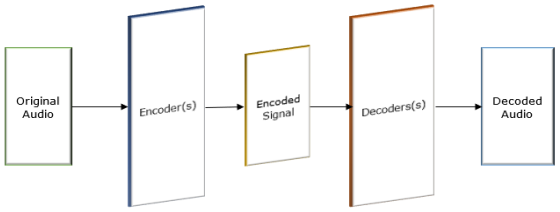

Figure 1: Generalized Autoencoder model

3.3.2 Convolutional Neural Networks (CNNs)

CNNs are utilized to extract hierarchical features from audio spectrograms. They facilitate the capture of local and global patterns in the audio data. The use of a CNN-based architecture enhances the robustness of audio fingerprints, even when subjected to noise and variations.

In Figure 1, encoder and decoder layers comprise of convolutional layers which are also used in CNNs.

3.4 Designing Autoencoder model: U-Net Autoencoder

The selection of the U-Net autoencoder architecture is driven by its potential to address the core objectives of our aim: creating audio fingerprints that are compact, informative, and resilient to noise.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

3.4.1 Configurations for the U-Net model

- epochs = 20 or 60 - If normalization of Mel Spectrogram is enabled, then number of model training iterations is set to 20, otherwise 60
- learning_rate = 1e-4 - determines the size of the steps taken during Adam optimization of the model parameters when updating the model weights.
- dropout_value = 0.3 - Regularization technique to prevent overfitting during model training

3.4.2 Architecture of the U-Net model

U-Net is a specialized variant of autoencoders, originally designed for image segmentation tasks [7]. Its name derives from its characteristic "U" shape, which reflects its unique architectural layout. U-Net consists of a compression path (encoder) and an expansion path (decoder), enabling the model to capture fine-grained details while preserving contextual information.

The encoder, through a series of convolutional layers, progressively reduces the spatial dimensions of the input, allowing it to extract hierarchical features. In contrast, the decoder employs transposed convolutional layers along with up sampling to upscale the feature maps, gradually reconstructing the original input's spatial dimensions. What distinguishes U-Net is its extensive use of skip connections, which connect corresponding layers of the encoder and decoder. These skip connections facilitate the fusion of features from multiple scales, enabling the model to capture both local and global information, while adding denoising capabilities [8].

These features are visualized in the Figure 2 and Figure 5. The original audio (containing 24000 samples for a 4 second audio at 48 kHz sampling rate) is encoded to a lower dimensional space containing 320 samples. This forms the fingerprint of the audio which can be utilized for further tasks.

Figure 3 and Figure 4 depicts the model as implemented for the work.

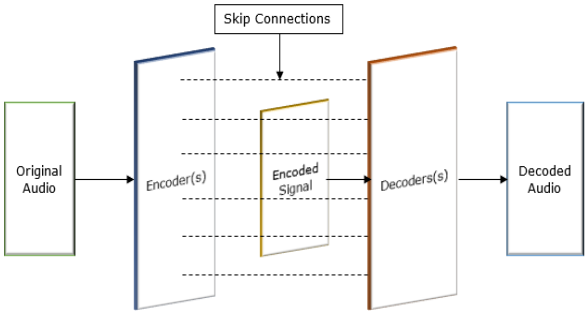

Figure 2: Generalized Autoencoder model modified to represent U-Net Autoencoder

```python
class UNetAutoEncoder(nn.Module):
    def __init__(self, in_channels=1, out_channels=1):
        super(UNetAutoEncoder, self).__init__()
        self.encoder = nn.ModuleList([
            EncodeLayer(in_channels, 64, norm=True),
            EncodeLayer(64, 128),
            EncodeLayer(128, 256),
            EncodeLayer(256, 256, dropout=Config.dropout_value),
            EncodeLayer(256, 256, dropout=Config.dropout_value),
            EncodeLayer(256, 64, dropout=Config.dropout_value)
        ])
        self.decoder = nn.ModuleList([
            DecodeLayer(64, 256, kernel_size=(2, 3), stride=2, padding=0, dropout=Config.dropout_value),
            DecodeLayer(512, 256, kernel_size=(2, 3), stride=2, padding=0, dropout=Config.dropout_value),
            DecodeLayer(512, 256, kernel_size=(2, 3), stride=2, padding=0, dropout=Config.dropout_value),
            DecodeLayer(512, 128, dropout=Config.dropout_value),
            DecodeLayer(256, 64)
        ])

        self.upsample_layer = nn.Upsample(scale_factor=2)
        self.zero_pad = nn.ZeroPad2d((1, 0, 1, 0))
        self.final_conv_layer = nn.Conv2d(128, out_channels, 4, padding=1)
        self.activation = nn.LeakyReLU(0.3, inplace=True)
```

Figure 3: Implementation of U-Net Autoencoder (1)

```python
class EncodeLayer(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=4, stride=2, padding=1, norm=True, dropout=0.0):
        super(EncodeLayer, self).__init__()
        self.layers = [
            nn.Conv2d(in_channels, out_channels, kernel_size, stride=stride, padding=padding),
            nn.LeakyReLU(0.2, inplace = True)
        ]
        if norm:
            self.layers.append(nn.InstanceNorm2d(out_channels))
        if dropout:
            self.layers.append(nn.Dropout(dropout))
        self.layers = nn.Sequential(*self.layers)

    def forward(self, x):
        return self.layers(x)

class DecodeLayer(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=4, stride=2, padding=1, dropout=0.0):
        super(DecodeLayer, self).__init__()
        self.layers = [
            nn.ConvTranspose2d(in_channels, out_channels, kernel_size=kernel_size, stride=stride,
                               padding=padding),
            nn.InstanceNorm2d(out_channels),
            nn.LeakyReLU(0.3, inplace=True)
        ]
        if dropout:
            self.layers.append(nn.Dropout(dropout))
        self.layers = nn.Sequential(*self.layers)

    def forward(self, x, encoded_input):
        x = self.layers(x)
        output = torch.cat((x, encoded_input), 1)
        return output
```

Figure 4: Implementation of U-Net Autoencoder (2)

As can be observed from Figure 3 and Figure 4, the encoder part of the model has 6 encoders, which progressively compacts the audio sample, so that the size of the encoded representation becomes torch.Size([1, 64, 1, 5]) (batch size: 1, number of channels: 64, height/frequency band: 1, width/ time steps: 5) as compared to the original audio size of torch.Size([1, 1, 64, 376]) (batch size: 1, number of channels: 1, height/frequency band: 64, width/ time steps: 376).

The decoder then reconstructs the audio from this encoded representation to the original size of torch.Size([1, 1, 64, 376]) via 5 decoder layers and 1 final decoding layer.

Each layer of the encoder and decoder uses LeakyReLU (Equation 1) as the activation function (described in section 3.5).

For regularization, the last 3 encoder layers and the last 2 decoder layers have a dropout value. This is to avoid overfitting of the model and to make the model more robust by randomly activating and deactivating some neurons. Also, normalization is applied in intermediate encoder layers on feature maps using InstanceNorm2D layer.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

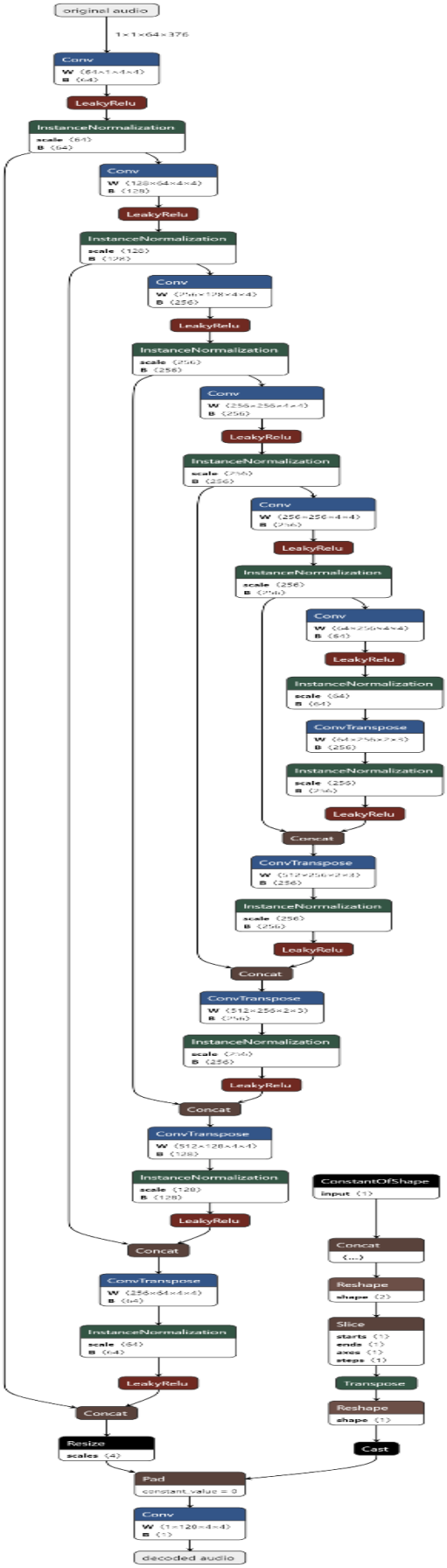

Figure 5: U-Net Autoencoder model used in the project

Figure 6 shows the trainable parameters of the model as well as forward and backward pass sizes:

```
==========================================================================
Total params: 6,001,601
Trainable params: 6,001,601
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 98.54
Params size (MB): 22.89
Estimated Total Size (MB): 121.53
--------------------------------------------------------------------------
```

Figure 6: Technical details of the model

3.4.3 Loss functions and Optimizations

The training of U-Net autoencoders involves optimizing a loss function, which measures the discrepancy between the reconstructed output and the original input. For our audio fingerprinting task, mean squared error (MSE) is used. During training, the model iteratively adjusts its weights using optimization algorithms like Adam to minimize the chosen loss function.

3.4.4 Challenges and Considerations

While U-Net autoencoders offer promising advantages, there are certain challenges in adapting them to the audio fingerprinting context. These challenges may include addressing data variability, managing model complexity, and mitigating overfitting.

- During the transformation of the data using Mel Spectrogram, the values of the frames are scaled logarithmically. Taking the logarithm of the Mel-filtered coefficients helps to approximate the logarithmic perception of loudness by the human ear [9]
- Mel spectrograms often undergo data normalization. One common approach is to subtract the mean and divide by the standard deviation along the frequency axis (i.e., across all frames at each frequency bin). This normalization helps in making the features more robust to variations in loudness and amplitude across different audio samples.
- Mel spectrograms help resolve issues of data variability by representing audio data in a way that is robust to slight variations in pitch, loudness, and timing. This is particularly useful for speech and audio classification tasks, where model needs to focus on the content and not be overly sensitive to variations in recording conditions

3.5 Training the model

The training of deep learning models is a critical step in creating effective audio fingerprints. The following steps are undertaken:

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

- Data splitting into training, validation and testing subsets.
  - Total number of audios in the dataset: 4736
  - Train dataset contains 70% of the audio samples
  - Validation dataset contains 15% of the remaining audio samples
  - Test dataset contains the rest 15% of the audio samples
- Model hyperparameter tuning, including layer configurations, activation functions, and loss functions.
  - The layer configurations are shown in Figure 3 and Figure 4
  - The primary activation function used is LeakyReLU (Equation 1), in all dense layers of encoder and decoder

$$f(x) = \begin{cases} x, if\ x > 0 \\ \alpha.x, if\ x \leq 0 \end{cases}$$

Equation 1: Function for LeakyReLU

In this equation:

- x represents the input to the activation function, an audio tensor in our case.
- α (usually a small positive constant, e.g., 0.2) is the "leak" coefficient that determines the slope of the function for negative inputs. It's a hyperparameter that can be adjusted based on the specific problem or data.
  - Loss function of MSE (Mean Squared Error) is used for training and validation loss. It calculates the mean squared error (squared L2 norm) between each of the elements in the input sample x (original audio tensor) and target sample y (decoded audio tensor). It finds how far is the reconstructed audio from the original audio
- Training iterations with backpropagation and optimization techniques such as Adam.
  - Training iterations is selected as 20 if normalization is applied for Mel Spectrogram, otherwise iteration is fixed at 60. This is because based on several runs, the losses converge and stabilizes at approximately iteration 20-25 and 60-65 respectively
  - As discussed in section 3.4.3, Adam optimization is used, with the learning rate as 1e-4
  - During training, the MSE loss is backpropagated which enables updating the weights between the dense layers

3.6 Testing the model

Once the model is trained, it is tested by following steps:

1. Test song is selected which is devoid of noise
2. Noise is added to this selected song
3. Both clean and noisy version of the song are preprocessed, and Mel spectrogram is generated. It is the first 4 seconds segment of the song
4. Encoder part of the model is called to generate fingerprint of this preprocessed song segment and saved. Further steps are to test reconstruction quality of the model
5. Call the test method, which calls the whole model process (encoder and decoder) for the clean and noisy audio generated from steps 3
6. After audio is decoded, Inverse Mel Spectrogram transformation is applied on it
7. This audio signal is then compared with original audio (clean version of song generated from step 3)
   a. MSE and Peak Signal-to-Noise ratio is calculated
   b. Test results and visualizations are described in Chapter 4

3.7 Creating audio database and searching using generated fingerprint

3.7.1 Data Information

A list of 235 songs in .wav format is chosen. For this work/project, following step was carried out:

- Convert songs into .wav format (if they are not already in .wav format)
  - Initially, all songs were in .mp3 compressed format. However, for consistency purposes, these should be converted to uncompressed .wav format as the model is trained on .wav files.
  - This design choice was taken so that irrespective of the compression of audio (mp3, mp4, etc.), the model would be correctly trained on a consistent wav version of the audio

Once the data (consisting of songs in wav format) is ready, it can then be used to create and search the audio database for music identification. This comprises of 2 tasks:

1. Create audio/song database

a) For each song, split it into segments of 4 seconds duration each (as the model is trained to generate fingerprints for 4 seconds duration of audio), with 1.5 seconds of overlap between each segment
b) Encode each segment using the trained model. This refers to the fingerprint of each segment
c) Insert these fingerprints of the segments into the database. This can be further achieved using 2 potential approaches:
   i. Store in a relational database

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

ii. Store in a vector database and dictionary

2. Query the fingerprint of the test clip against the audio database

Details of each step are in the following subsections.

3.7.2 Create Audio Database

Once the encoded segments (fingerprints) are ready, they are inserted into database.

3.7.2.1 Storing in a Relational Database

One of the approaches is to store the encoded segments into a relational database (or a NoSQL database).

Following steps would have to be carried out for this:

1. Encoded segment will be a Tensor datatype
2. Flatten this tensor into 1D list
3. This information can then be stored into a relational/ NoSQL database table in following possible ways:
   a) Store the list of values (from step 2) in serialized form. While querying, L2 distance of each list will be calculated against the queried fingerprint. The one with least L2 distance would be returned. Also, the song name, songId, segmentId would be stored
      i. Challenge – While querying, this would require traversing each list of fingerprints and calculating L2 distance. This would take a considerable amount of time in case of large number of songs
   b) Generate a 5-point summary of this list (mean, median, 25th quartile, 50th quartile, 75th quartile) and note down the offset values of top 5 values of the list. Also, the song name, songId, segmentId would be stored
      ii. Challenge – While querying, the threshold to use for 5-point summary and offsets for test audio fingerprint would be hard to determine. As the values for the queried fingerprint might defer considerably from the ones stored in database table for the expected song

Owing to above challenges, storing the records in vector database was decided.

3.7.2.2 Storing in a Vector Database – Preferred Approach

This work/project employs a vector database to construct an audio database. Vector databases are specialized databases optimized for storing and retrieving high-dimensional data vectors. Vectors are mathematical objects that represent magnitudes and directions, commonly used to represent data in machine learning applications [11].

Vector databases offer a highly efficient search functionality for retrieving closest matching vectors. They can also store, search and retrieve vectors of any dimension [11], and supporting incremental updates, making them an ideal choice for storing audio fingerprints in vectorized form. Facebook AI Similarity Search (FAISS) is one such vector database.

FAISS is used in variety of tasks which involves working with tensors/ vectors like image search, retrieve closest matching vectors against a query, etc. [6].

FAISS works by building an index of the vectors in the database. This index allows FAISS to quickly and efficiently find the vectors that are most similar to a query vector. It can also retrieve ‘k’ closest matching vectors. FAISS provides a variety of different indexing algorithms, each with its own trade-offs between speed and accuracy [6].

For our use case, hybrid index is used, with a combination of Map and Flat indexes (Figure 7):

```
d = 320
audio_index = faiss.IndexIDMap(faiss.IndexFlatL2(d))
```

Figure 7: FAISS Index initialization

1. IndexIDMap: This index encapsulates another index and translates IDs when adding and searching. It maintains a table with the mapping, which can be used to insert IDs along with vectors.
2. IndexFlatL2: This index is a simple linear scan index that stores the vectors in memory and searches for similar vectors by computing the L2 distance between the query vector and all the vectors in the database.

As we are using hybrid indexes, we can store song IDs along with the vector of each segment of a particular song. The corresponding song name for each song ID is maintained in a separate data structure (Python dictionary object).

The pseudocode in the Figure 8 describes the flow of adding the song vectors along with song ID in index, while maintaining a dictionary to map song names with song ID.

The pseudocode also handles incremental updates to the index. As Flat index (IndexFlatL2) is being used in the work, re-training and re-indexing is not required. However, to maintain the correct IDs for the vectors and dictionary, the highest ID is retrieved from the index and used for new data additions.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

```
function add_songs_in_index(audio_index, song_name_dict, songs_directory,
segment_duration, overlap, model):
    song_id = get_max_id_from_index(audio_index) + 1

    for each song_file in list_files_in_directory(songs_directory):
        start_time = 0
        end_time = segment_duration
        song, sample_rate = load_audio(song_file)
        song_duration = calculate_song_duration(song, sample_rate)

        if segment_duration > song_duration:
            continue

        segment_start = 0
        while segment_start + segment_duration <= song_duration:
            segment_end = segment_start + segment_duration
            segment = extract_audio_segment(song, sample_rate,
segment_start, segment_end)
            segment_fingerprint = extract_features_and_encode(segment,
model)
            encoded_segment_vector = vectorize(segment_fingerprint)
            M, N = shape_of(encoded_segment_vector)

            //Create a vector of song_id which has the same first
           dimensional shape as that of encoded_segment_vector
            id_vector = [song_id for _ in range(M)]

            audio_index.add_with_ids(encoded_segment_vector, id_vector)

            segment_start += (segment_duration - overlap)
        endwhile
        song_name_dict[song_id] = get_song_name(song_file)
        song_id += 1
     endfor
    return audio_index, song_name_dict
```

Figure 8: Pseudocode to incrementally add songs in FAISS index

3.7.3 Query the created audio database

1. For the test audio, preprocess it as per section 3.2
2. Then generate fingerprint of this preprocessed test audio signal by calling the encode method of the trained U-Net Autoencoder model.
   a. Returned shape is a tensor of size (64, 1, 5)
3. Flatten this encoded fingerprint and generate vectorized version of the fingerprint.
   b. Final shape would be a vector of size (1, 320)
4. Query the vector database using this vector fingerprint
   a. Number of closest matching records (k) should be determined.
   b. For work/project purposes, k is set to 5. This implies that 5 closest matching songs to the queried fingerprint are retrieved
5. For testing purposes, a random 4 second segment of a random song (from the list of 235 songs) was selected and queried against the created audio index
6. The output of the search is described in section 4.4.

IV. RESULTS

This section depicts the results of the U-Net Autoencoder model created as per previous chapter, and the outputs of the audio search logic.

4.1 Train and validation loss after training the model

Losses after 20 iterations of training:

```
-------------------------
[Epoch 20] Training Loss: 0.14785
[Epoch 20] Validation Loss: 0.15930
-------------------------
```

Figure 9: Training and validation loss after 20 iterations

4.2 Testing the trained model, Audio quality metrics and samples

After training, the model is tested with test dataset in evaluation mode. Running the model in evaluation mode is essential when we want to ensure that the model is in the correct mode for inference or evaluation, without any training-specific alterations like dropout or batch normalization.

The metrics used to determine the effectiveness of the model in reconstructing the audio from compressed, encoded fingerprint is measured using MSE (Mean Squared Error) and PSNR (Peak Signal-to-Noise Ratio).

For a good reconstruction, there should be as less MSE as possible, and PSNR should in range of 40-60 dB [10].

For the test dataset, the value for MSE is 0.4976 and PSNR is 51.162.

```
1 original_test_clean_audio_clip, original_test_noisy_audio_clip, decoded_test_audio_clip = test(test_audio_clip_dataloa

Test input shape: torch.Size([1, 1, 64, 376])
Test decoded shape: torch.Size([1, 1, 64, 376])
For the test data - Mean Squared Error (MSE): 0.49761, Mean Peak Signal-to-Noise Ratio (PSNR): 51.16195
```

Figure 10: Performance metrics for test data

4.3 Visualizations of the outputs

After training and testing the model, waveforms for original audio, encoded audio fingerprint and decoded audio were generated.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

4.3.1 Plots of Waveforms

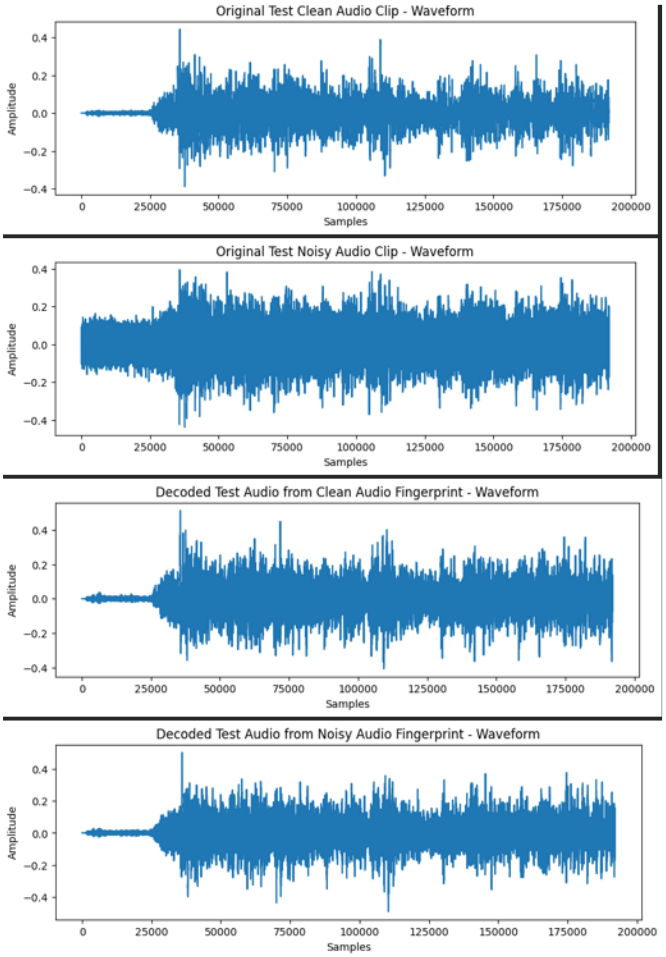

Figure 11: Plots of Waveforms

These are the plots of the audio waveforms of 4 seconds duration with 48 kHz sampling rate, with x-axis denoting the samples in the audio data, and y-axis denoting the amplitude of the signals. First plot depicts the waveform of the original audio sample devoid of noise. The second plot is the waveform of the noisy version of the clean audio sample. The third plot shows the waveform of the decoded audio from the fingerprint of clean audio sample. And the fourth plot depicts the decoded audio from the fingerprint of the noisy audio sample.

The decoded plots of both clean and noisy audio samples bear a close resemblance to that of original clean audio sample. This shows that there's a very close matching between the original and the decoded audio waveforms, with some fluctuations in the amplitude; and proves that the fingerprint generated is noise resilient.

Summarizing the above observation - the generated encoded representation of the audio (fingerprint) is robust so that the model can closely reconstruct the audio from the fingerprint albeit of noise.

4.3.2 Plots of Mel Spectrogram

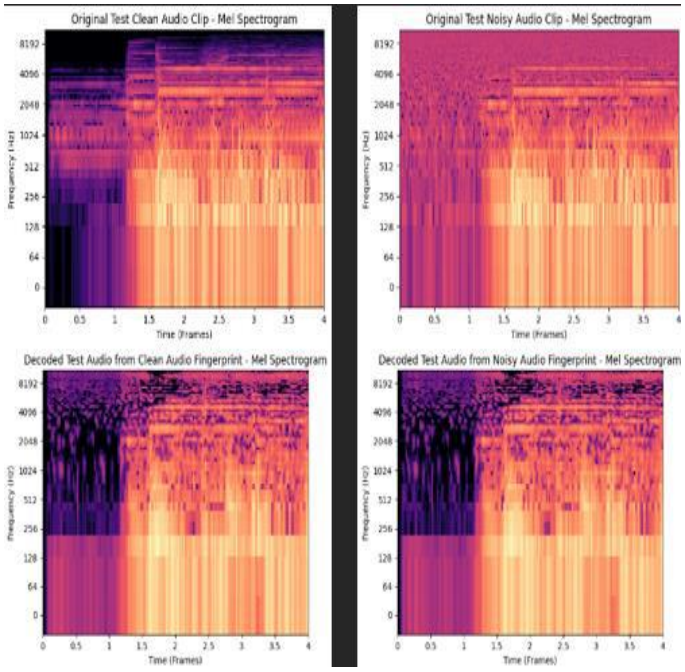

Figure 12: Plots of Mel Spectrogram

These are the plots of the Mel Spectrogram of the clean and noisy audio signals, along with the corresponding decoded audio signals from respective fingerprints of the original audio signals. The x-axis denotes frames of the audio and y-axis denoting frequency in logarithmic scale.

From the plots, the frequencies for decoded signals have a minor variance with that of the original signal. But overall, this plot also suggests a close resemblance between the original and decoded audio signals, which again proves the robustness of the encoded fingerprint (from which the decoded signal is generated), even in the presence of noise.

The audio samples and metrics from previous section and above plots denote the correctness of the model in generating fingerprint and reconstructing the audio from that fingerprint. This also suggests that this generated fingerprint is robust.

4.4. Output of searching the audio vector database

A random song is chosen from a list of 235 songs (which were used for creating audio vector database), and a random 4 second segment is extracted.

4.4.1 Song segment without noise

The segment chosen is then encoded and passed into the model to generate an encoded fingerprint. It is then vectorized which can be used for querying the created audio vector database.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 9, Issue 3, pp 182-192, March-2025

https://doi.org/10.47001/IRJIET/2025.903024

This is the output:

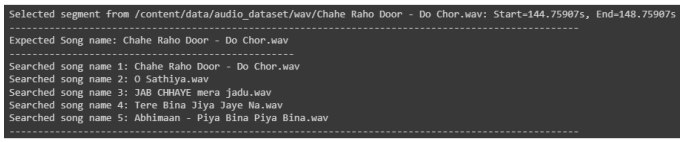

Figure 13: Output of the audio search from FAISS vector database without noise in the segment

4.4.2 Song segment with noise

Also, the same search was carried out by adding some noise in the audio. Steps followed were:

1. Add noise to the chosen audio segment
2. Generate encoded fingerprint of this noisy audio segment
   a. Tensor of shape (64, 1, 5) is returned
3. Flatten and vectorize it into shape (1, 320)
4. Use this vector to query the database
5. Output:

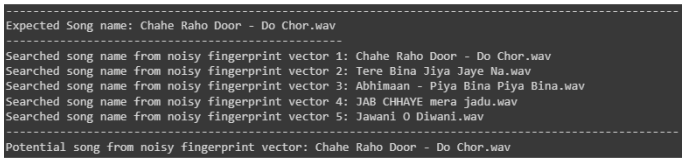

Figure 14: Output of the audio search from FAISS vector database with noise in the segment

V. SUMMARY/ CONCLUSIONS

The model using U-Net Autoencoder is developed, which generates robust and noise-resilient audio fingerprints. Also, the logic to create audio database using vector database and querying it for the test fingerprint is developed.

These developments fulfill all the objectives for the work/project, which is to generate robust audio fingerprints and use them to find the closest matching audio from the audio database; and this paper delves into the details on how these are achieved.

A general introduction about the aim of this work/project and paper, as well as the scope of the work is described in Chapter 1.

The paper goes into details about various researches in this field, and their shortcomings in Chapter 2.

Chapter 3 in the paper proposes the methodology to be used for generating the audio fingerprints using U-Net Autoencoder and decoding the audio from the fingerprints (of both clean and noisy audio samples) and comparing it with the original audio sample. The details on the U-Net Autoencoder model used for this purpose are described from Chapter 3 – Section 3.4 onwards.

Chapter 3 – Section 3.7 further describes the potential ways to create audio database and querying the database to retrieve song name based on the encoded fingerprint of audio. It discusses on various approaches which can used to create song database. It also then delves further into shortcomings of one method of storing in relational database. To overcome the shortcomings, approach using vector database is described, specifically FAISS database. It provides a pseudocode of the addition of fingerprints of the songs in the index of FAISS vector database and the logic of querying the test fingerprints from this index.

Chapter 4 describes the results of the implementation of the proposed methodology, with the MSE of decoded audio being 0.4976 and PSNR being 51.1132 (section 4.1 and 4.2) which suggests a very good reconstruction of the audio from encoded signal (fingerprint). The comparison of the waveform and Mel spectrogram heatmaps of the reconstructed audio from clean and noisy versions of the original audio is plotted in section 4.3. Further, the outcome of the search of test audio fingerprint in the database is given in section 4.4.

As a part of the future work, the implementations can be exposed via APIs which can be integrated with applications such as Music Recognition systems or Content Management Systems. Additionally, exploring the optimization of model efficiency for large-scale data can further enhance practical usage.

ACKNOWLEDGEMENT

I sincerely thank my mentors, Mr. Shashank Khasare (Infosys Ltd.) and Mrs. Sneha Dixit (Infosys Ltd.), for their invaluable guidance on this topic. Their insights and support enabled me to explore the techniques and develop a solution for encoding, decoding and identifying the audio using fingerprints.

REFERENCES

[1] Avery, Li-Chun, Wang, “An industrial-strength audio search algorithm”, 582-588, doi: 10.5072/ZENODO.243872, 2004.

[2] Haitsma, Jaap & Kalker, Ton, “A Highly Robust Audio Fingerprinting System”, Proc Int Symp Music Info Retrieval 32, 2002.

[3] P. Panyapanuwat, S. Kamonsantiroj and L. Pipanmaekaporn, "Similarity-preserving hash for content-based audio retrieval using unsupervised deep neural networks”, International Journal of Electrical and Computer Engineering 11.1, 879, 2021.

ISSN (online): 2581-3048
Volume 9, Issue 3, pp 182-192, March-2025
https://doi.org/10.47001/IRJIET/2025.903024

[4] Choi, K., Fazekas, G., Cho, K. and Sandler, M., "A tutorial on deep learning for music information retrieval", arXiv preprint arXiv:1709.04396, 2017

[5] Valentini-Botinhao, Cassia, "Noisy speech database for training speech enhancement algorithms and TTS models", University of Edinburgh. School of Informatics. Centre for Speech Technology Research (CSTR). https://doi.org/10.7488/ds/2117, 2017.

[6] Johnson, J., Douze, M., & Jégou, H., "Billion-scale similarity search with GPUs", ArXiv. /abs/1702.08734, https://github.com/facebookresearch/faiss, 2017

[7] Ronneberger, O.; Fischer, P.; Brox, "U-net: Convolutional networks for biomedical image segmentation", In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, Springer: Cham, Switzerland, 5–9 October 2015

[8] M. Tripathi, "Facial image denoising using AutoEncoder and UNET", Heritage and Sustainable Development, vol. 3, no. 2, pp. 89–96, Oct. 2021.

[9] "Lyra: A New Very Low-Bitrate Codec for Speech Compression", ai.googleblog.com 25 February 2021

[10] Chervyakov, N.; Lyakhov, P.; Nagornov, N., "Analysis of the Quantization Noise in Discrete Wavelet Transform Filters for 3D Medical Imaging". Appl. Sci,, 10, 1223, 2020, https://doi.org/10.3390/app10041223

[11] Amazon AWS. Vector Database - https://aws.amazon.com/what-is/vector-databases/

Citation of this Article:

Divesh Singh. (2025). Deep Learning-based Fingerprinting Methods for Audio Representation and Search. *International Research Journal of Innovations in Engineering and Technology - IRJIET*, 9(3), 182-192. Article DOI https://doi.org/10.47001/IRJIET/2025.903024

****

