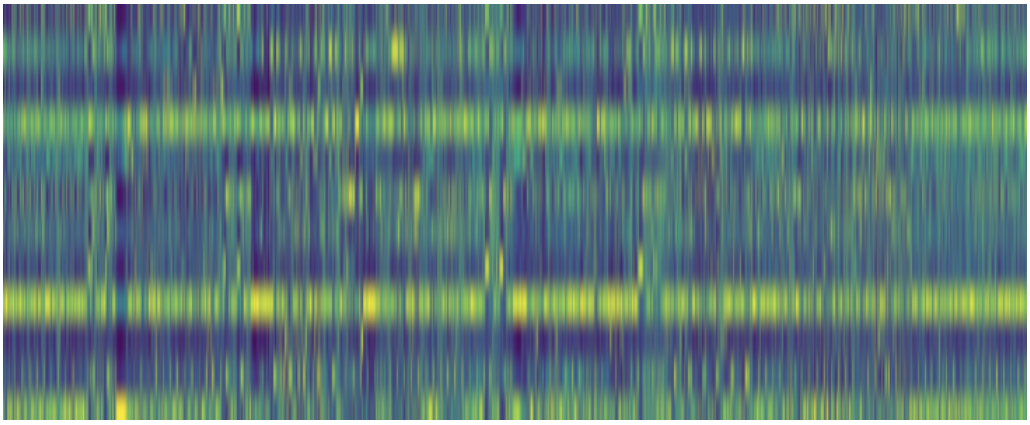

Cover song Similarity: A Deep Learning Approach

Author:
Candidate Athanasios Aidinis
email: thanos.aidinis@gmail.com

December 25, 2024

Contents

1 Objective **3**

2 Limitations **3**

3 Dataset Description **3**
3.1 Small-Scale Data Analysis . . . 3
3.2 Why HPCP? . . . 4

4 Data Preprocessing **5**
4.1 Normalization . . . 5
4.2 Gaussian Noise Addition . . . 6
4.3 Zero Padding . . . 7
4.4 Creating Pairs of Similar and Dissimilar Data . . . 7

5 Data Augmentation with Pitch Shifting **8**

6 Model Architecture **8**
6.1 Model Architecture and Loss Functions . . . 9

7 Experiments & Results **10**
7.1 Model Training . . . 10

8 Usability & Scalabilty **11**

9 Conclusions **11**

List of Figures

1 Pie chart about title similarity. . . . 4
2 Example of HPCP plot showing harmonic content over time. . . . 5
3 Example of normalized HPCP plot. . . . 6
4 Example of HPCP plot with added Gaussian Noise. . . . 6
5 Simple example of Data Augmentation. . . . 8
6 Example of SNN Architecture [3]. . . . 9
7 Loss and Accuracy during 6 epochs of training. . . . 10
8 Accuracy scores for Train, Validation and Test set. . . . 11
9 FastAPI, Celery, Redis architecture overview [5]. . . . 11

Abstract

This report explores the development of a similarity metric for cover song detection using a Siamese Convolutional Neural Network (CNN). The model is trained to minimize the distance between the embeddings of the original and cover songs while maximizing the distance for unrelated tracks. Using HPCP features from the Da-TACOS dataset, the proposed approach demonstrates the effectiveness of deep learning techniques in capturing tonal similarities and achieving robust performance in cover song detection.

Keywords: Cover Song Detection, Siamese CNN, HPCP Features, Similarity Metric, Deep Learning, Da-TACOS Dataset, MIREX

1 Objective

The primary objective of this task is to design and implement a similarity metric that quantifies the resemblance between original songs and their cover versions. The metric should:

- Minimize the distance for song pairs with a cover relationship.
- Maximize the distance for unrelated song pairs.

This work evaluates the effectiveness of the proposed metric (as a proof-of-concept) and provides insights into its limitations and potential for future improvements.

2 Limitations

Several limitations impacted the scope and implementation of this task:

- **Time Constraints**: The task was completed in one week, alongside another assignment (Text Normalization), limiting the extent of experimentation and refinement.
- **Hardware Limitations**: Due to computational constraints, only a proportion of the data set was used and certain simplifications were made. These simplifications will be discussed in later sections.

3 Dataset Description

The dataset utilized for this task is the Da-TACOS benchmark subset [1]. The subset is particularly suitable due to its:

- **Standardized Evaluation**: Designed for evaluating cover detection systems, aligning directly with this task.
- **Reproducibility**: Ensures results can be compared with previous work.
- **Scale**: The dataset contains 15,000 songs, offering ample data for model training, validation, and testing.

The dataset includes metadata such as title and artist, along with pre-extracted features like HPCP and Chroma. For this task, HPCP was chosen due to its robustness to key shifts and proven effectiveness in cover detection tasks. An example can be found in the following table 1.

Table 1: Example of Metadata in the Dataset

Field	**Value**
Work Title	Light My Fire
Work Artist	John Densmore
Performance Title	Light My Fire
Performance Artist	The Doors
Release Year	1967
Work ID	W_22
Performance ID	P_22
Instrumental	No

3.1 Small-Scale Data Analysis

To better understand the dataset, a small-scale analysis was conducted.

- **Title Matching**: Approximately 84. 65% of the cover songs retain the original title. Specifically, 12,698 of 15,000 songs share the same title between the original and cover versions. This highlights a significant trend in title retention (see Figure 1).

- **Artist Matching**: Only 300 of 15,000 songs have the same artist listed for both the original and the cover, indicating a low overlap.

Additional text-based analyzes [2] could be performed to assess title similarity using methods such as TF-IDF cosine similarity. Furthermore, preprocessing steps can remove title invariances introduced by descriptors like "live," "cover," "remix," etc., to improve detection robustness. Metadata quality, such as artist credits, may also impact results and should be explored in future work.

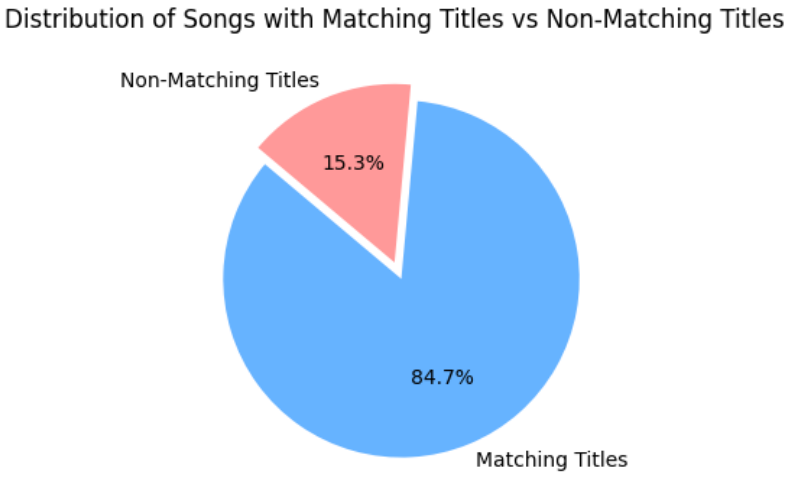

Figure 1: Pie chart about title similarity.

3.2 Why HPCP?

HPCP features are chosen for their robust performance in cover song detection for the following reasons. Firstly, **HPCP is robust to key shifts**, as it normalizes pitch classes across octaves and can adjust for transpositions more effectively than Chroma features. Secondly, it demonstrates **timbre independence**, focusing on tonal similarity while ignoring instrumental or vocal timbre, which is crucial for identifying cover songs. Lastly, HPCP has a proven track record , having been widely used and benchmarked for tasks such as the detection of cover songs, often outperforming simpler features like Chroma [2].

Although the chroma features provide useful tonal information, they lack the robustness of HPCP for this task. Using both features might increase redundancy without significantly improving performance. Instead, complementary features such as rhythm or tempo-related metrics could be explored to enhance detection accuracy [2].

The HPCP plot showcases harmonic activity over time, where the x-axis represents time frames and the y-axis corresponds to the 12 pitch classes like C, C#, and D. Bright areas indicate stronger harmonic activity, making HPCP useful for identifying chord changes and repeated sections. Comparing these patterns across tracks reveals similarities crucial for cover song detection. (see Figure 2).

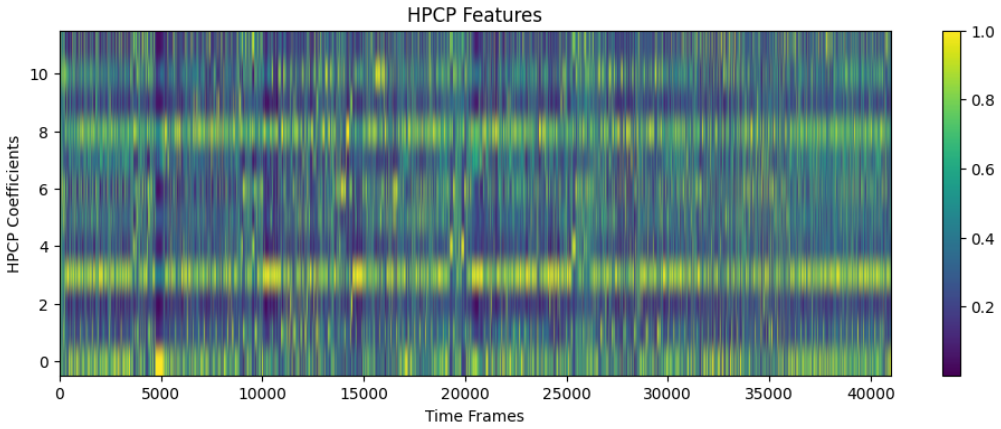

Figure 2: Example of HPCP plot showing harmonic content over time.

4 Data Preprocessing

For the preprocessing of the data, several techniques were employed to ensure the model's input was suitable for training. These steps include normalization, the addition of Gaussian noise, and zero-padding. Each step is detailed below.

4.1 Normalization

Normalization is essential to scale the feature values within a specific range, ensuring that the model's training process is more stable and efficient. Min-Max normalization was applied to the HPCP features, which transforms the values to a range between 0 and 1. The equation for Min-Max normalization is:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

where:

- x represents the original HPCP feature value,
- x_{min} is the minimum value of the feature,
- x_{max} is the maximum value of the feature,
- x_{norm} is the normalized feature value.

This normalization step ensures that all features have the same scale, preventing certain features from dominating the learning process due to their larger values (See Figure 3).

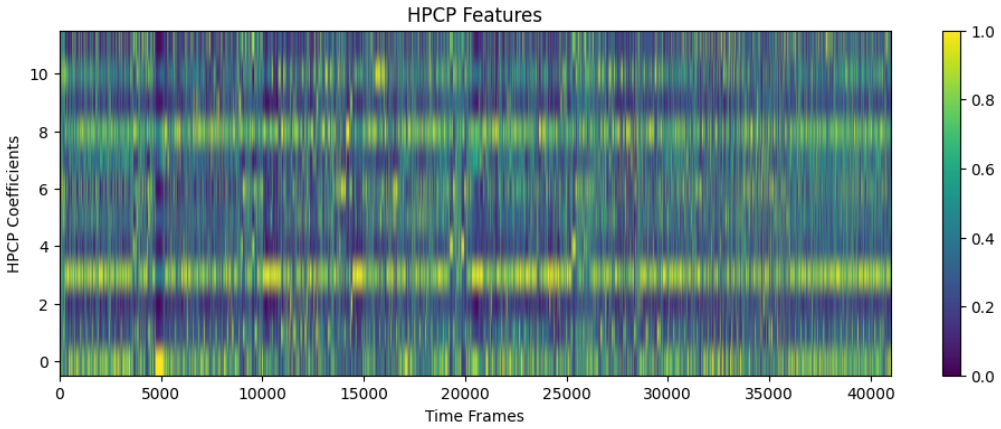

Figure 3: Example of normalized HPCP plot.

4.2 Gaussian Noise Addition

To increase the robustness of the model and help prevent overfitting, Gaussian noise was added to the normalized HPCP features. This noise simulates slight variations in the data and encourages the model to learn more generalized patterns. The Gaussian noise N is defined by the equation:

$$x_{\text{noisy}} = x_{\text{norm}} + \mathcal{N}(0, \sigma^2)$$

where:

- x_{norm} is the normalized feature value,
- $\mathcal{N}(0, \sigma^2)$ represents a Gaussian distribution with a mean of 0 and a variance of σ^2,
- x_{noisy} is the noisy feature value.

In this process, σ^2 is a parameter that controls the magnitude of the noise. Further experimentation can be done to finetune this paramater. The added noise helps the model generalize better, particularly in scenarios where the dataset might not capture all possible variations in the data (See Figure 4).

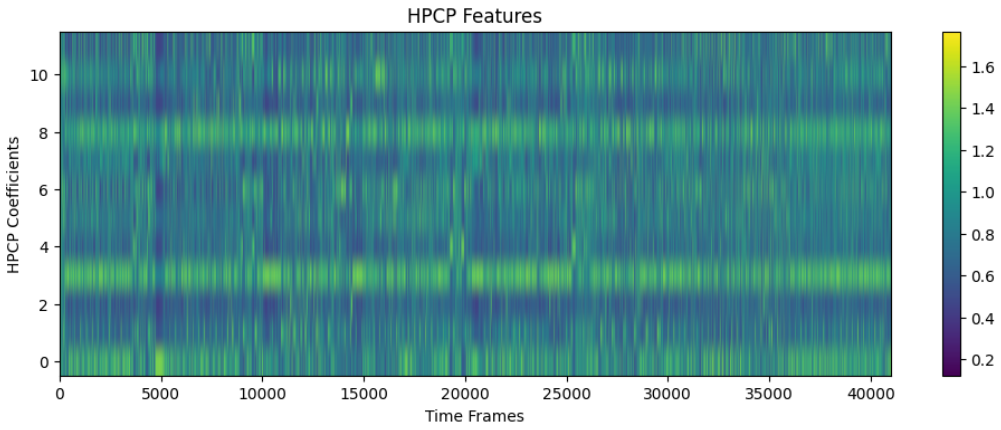

Figure 4: Example of HPCP plot with added Gaussian Noise.

4.3 Zero Padding

Zero padding is applied to the HPCP feature sequences to ensure uniformity in the length of all input sequences. Zero padding ensures that the model can process all input sequences in a consistent manner, regardless of their original length. This technique also helps preserve the temporal structure of the data, which is important for learning patterns across time. The equation for zero-padding can be expressed as:

$$x_{\text{padded}} = \left[\underbrace{0, 0, \ldots, 0}_{p_{\text{front}}}, x_1, x_2, \ldots, x_n, \underbrace{0, 0, \ldots, 0}_{p_{\text{back}}} \right]$$

where:

- $x_1, x_2, \ldots, x_n$ are the original HPCP feature values,
- p_{front} and p_{back} represent the number of zeros added at the front and back of the sequence, respectively.

For this project data was zero padded at the back.

4.4 Creating Pairs of Similar and Dissimilar Data

In this step, the preprocessed data is used to create pairs of similar and dissimilar samples for training the Siamese network. The objective is to generate a balanced set of pairs with both positive (similar) and negative (dissimilar) relationships. This is essential for the network to learn the similarity function between pairs of samples.

The 'create_pairs' method is responsible for generating these pairs. It takes as input the preprocessed data and the percentage of dissimilar pairs required.

- **Similar pairs (positive samples)**: These pairs consist of two samples that belong to the same class or category. For example, in the context of cover song detection, this would involve pairs of songs from the same artist or similar musical style.
- **Dissimilar pairs (negative samples)**: These pairs consist of two samples from different classes. In the cover song detection task, this would involve pairs of songs that are not covers of each other.

The process works as follows:

1. For each song in the dataset, pairs are created either from the same class (positive pairs) or from different classes (negative pairs).
2. The percentage of dissimilar pairs can be controlled by the `dissimilar_percentage` parameter. This allows fine-tuning of the number of dissimilar pairs relative to similar pairs. The dissimilar pairs are created in a random manner.
3. The pairs and their corresponding labels (0 for similar, 1 for dissimilar) are stored in two separate lists: `pairs` and `pair_labels`.

The pairs are saved in the form of pickle files, with the following two files created:

- `pairs.pickle`: Contains the list of pairs.
- `pair_labels.pickle`: Contains the corresponding labels for the pairs (1 for similar, 0 for dissimilar).

5 Data Augmentation with Pitch Shifting

In the realm of audio signal processing, data augmentation plays a pivotal role in enhancing the robustness and generalization capability of machine learning models. Augmentation techniques aim to diversify the training dataset by introducing variations in the input data without altering its inherent semantic meaning. Among various augmentation methods, pitch shifting stands out as a fundamental approach to manipulate audio signals while preserving their structural characteristics. A simple and high level example of data augmentation can be seen in the figure 5 below.

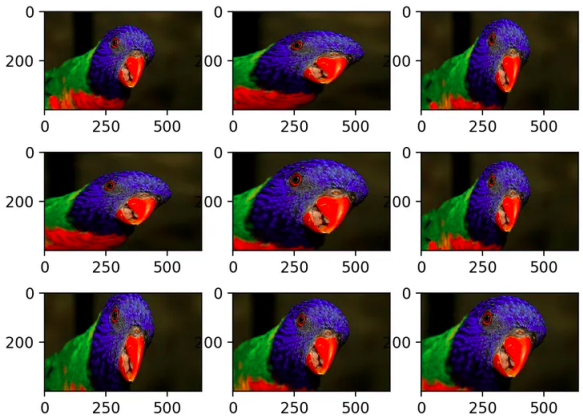

Figure 5: Simple example of Data Augmentation.

Pitch shifting involves altering the frequency content of an audio signal, thereby modifying its perceived pitch without affecting its temporal duration. This technique is commonly used in music production and audio processing to achieve desired musical effects, such as transposing melodies or harmonizing vocal lines. In the context of data augmentation, pitch shifting offers a versatile tool for generating augmented samples with varying pitch levels while retaining the original harmonic structure of the audio.

However, it should be noted that applying pitch shifting to HPCP data might not be a good idea. HPCP features, by design, capture harmonic pitch class profiles and are relatively invariant to key and pitch shifts. Introducing pitch shifting on the feature level could lead to the alteration of these profiles, potentially distorting the harmonic structure of the features. Since HPCP is designed to capture tonal similarity while being robust to key shifts, applying pitch shifting to the raw feature data may not yield meaningful or effective augmented samples, as it could disrupt the underlying tonal patterns that the model relies on for cover song detection.

Moreover, due to **hardware and time limitations**, this data augmentation technique was not explored further in this project. Given the computational constraints, it was prioritized to focus on the most effective preprocessing steps, such as normalization, noise addition, and zero padding, rather than experimenting with pitch-shifted HPCP features.

In future work, if computational resources and time allow, further investigation into pitch shifting of raw audio data or alternative augmentation methods for HPCP could be considered to evaluate their impact on model performance.

6 Model Architecture

Siamese networks, first introduced by Bromley et al. (1994) for signature verification, have become a powerful tool for learning similarity representations. These networks consist of two identical sub-networks that share the same weights and are trained to compare two input items, learning a feature representation that captures the relationship between them. In the context of cover song detection, the task is to determine whether two songs are different versions (i.e., covers) of the same original song. Siamese networks are well-suited for this problem because they can

learn discriminative features that capture the subtle similarities and differences between audio pairs, which is crucial for identifying cover songs. An example of the basic SNN architecture can be found in the figure 6

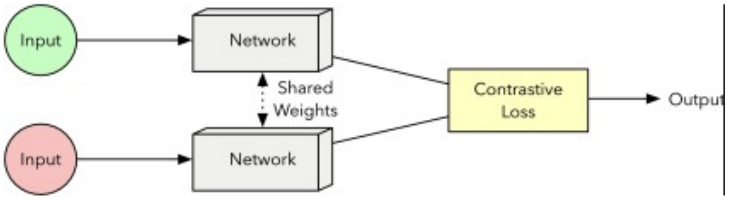

Figure 6: Example of SNN Architecture [3].

The key advantage of using Siamese networks in this domain lies in their ability to directly compare pairs of songs, providing a natural way to assess the similarity between them. Instead of training the model to classify individual songs, I train it to learn a similarity metric, which makes it particularly effective for tasks like cover song detection, where variations in pitch, arrangement, and instrumentation can obscure direct feature-based comparisons. This approach allows the network to focus on learning the inherent relationships between songs, irrespective of the exact differences in their features.

In this project, as proposed by Stamenovic (2018) [4], I chose Siamese networks because they allow us to effectively model the similarity between cover song pairs. By using convolutional layers to extract time-frequency representations of audio, the network learns to distinguish between songs that are covers and those that are not, with promising results. The architectures ability to learn useful representations for song comparison, without relying on hand-crafted features, aligns well with the complexities of the cover song detection problem, making it an ideal choice for this task.

6.1 Model Architecture and Loss Functions

The Siamese network designed for cover song detection consists of two identical sub-networks that process two input songs in parallel. These sub-networks share weights and are responsible for extracting feature representations from each song's time-frequency representation. The network architecture includes multiple layers, such as convolutional layers, max-pooling layers, dropout layers, and fully connected layers. These layers aim to learn discriminative features that can capture the similarity between the input song pairs.

The base network starts with a series of 1D convolutional layers, followed by max-pooling layers to reduce dimensionality and dropout layers for regularization. At the end of the network, fully connected layers transform the learned features into a compact representation, with the final output being a 64-dimensional vector.

Once the feature vectors for both input songs are extracted, a custom layer computes the cosine similarity between them. Then we get the 1-cosine similarity score to adjust similarity range to [0,1] where 0 is the similar and 1 the dissimilar.

$$1 \text{ - cosine\_similarity}(x_1, x_2) = 1 - \frac{x_1 \cdot x_2}{\|x_1\|\|x_2\|}$$

where x_1 and x_2 are the feature vectors for the two input songs. This similarity score is then used to determine whether the two songs are similar enough to be considered a cover song pair. Since cover songs share many musical characteristics despite differences in arrangement, cosine similarity provides an effective measure for detecting these relationships.

To train the network, we employ a contrastive loss function, which is commonly used in Siamese networks to learn a similarity metric. The contrastive loss function is defined as:

$$\mathcal{L}_{\text{contrastive}}(y, \hat{y}) = (1 - y) \cdot \hat{y}^2 + y \cdot \max(\text{margin} - \hat{y}, 0)^2$$

This loss function ensures minimization of loss if the similarity score predicted is close to the actual relationship of the two songs. For example, if for two similar songs (having 0 true score) the model outputs a similarity score of 0.01 then the loss is close to 0.

7 Experiments & Results

It is important to note that the following experiments are primarily conducted as proof of concept. Due to time constraints, no extensive fine-tuning or optimization processes were applied. Furthermore, only 50% of the HPCP dataset provided by DaTacos [1] was utilized in these experiments. Additionally, the number of pairs created for training is only a small fraction of the total potential pairs that could be generated from the available data. Therefore, the results presented here should be interpreted within the context of these limitations.

Initially, the possibility of leveraging a pre-trained encoder for the HPCP data was explored. However, after an extensive search, no suitable pre-trained models were found. As a result, I decided to train a custom encoder using convolutional layers, which demonstrated promising results. While this approach was effective, it is clear that the model's performance could be further improved with more extensive fine-tuning.

The following experiments were conducted to evaluate the performance of the Siamese network model on HPCP data. The model was trained for 6 epochs (After 6 epochs overfitting was occurring), with the training process and results summarized below.

7.1 Model Training

The model underwent six epochs of training, with each epoch achieving increasing accuracy and decreasing loss on both the training and validation datasets. The training process was computationally intensive, with each epoch taking approximately 213 seconds.

The accuracy is calculated by having a threshold of 0.5 on the similarity score, anything below is considered similar and anything above that threshold is considered dissimilar.

Data was split into training, validation and test set with percentages 60 %, 25% and 15% accordingly using a stratified split.

As seen in the figure below 7, the training accuracy steadily increased, with the model achieving 95.32% accuracy by the end of the 6th epoch. However, the validation accuracy showed less improvement, stabilizing around 70% by the final epoch.

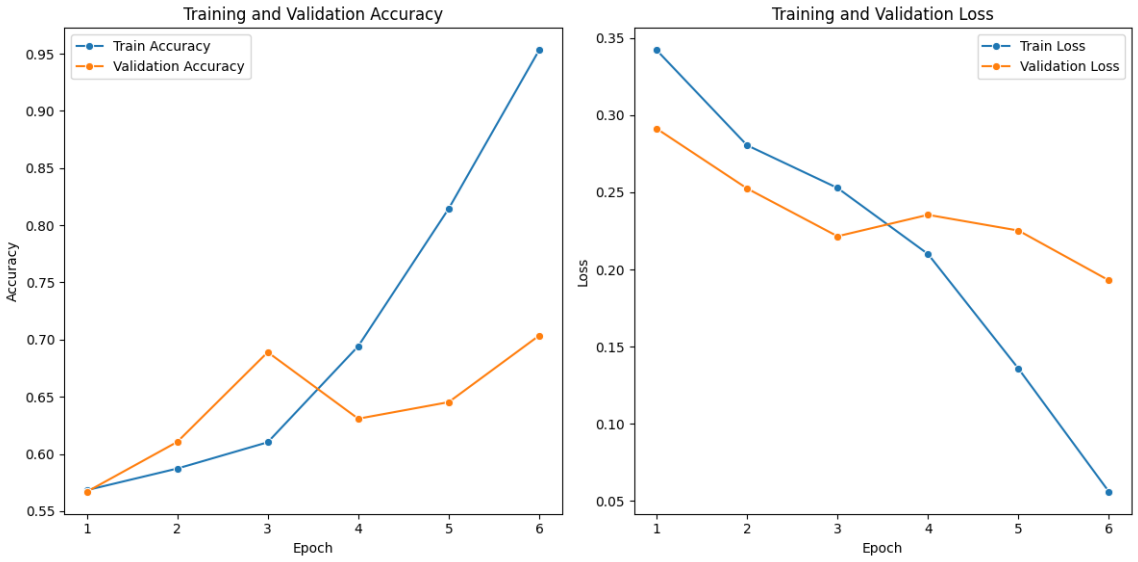

Figure 7: Loss and Accuracy during 6 epochs of training.

After training, the model's performance was evaluated on the test set. The accuracy of the model on the test set was reported as 68.85%, indicating that while the model performed well on the training set, there was some overfitting, as indicated by the relatively lower validation and test accuracies. The accuracy scores for the train, validation and test set can be seen in the barchart below 8

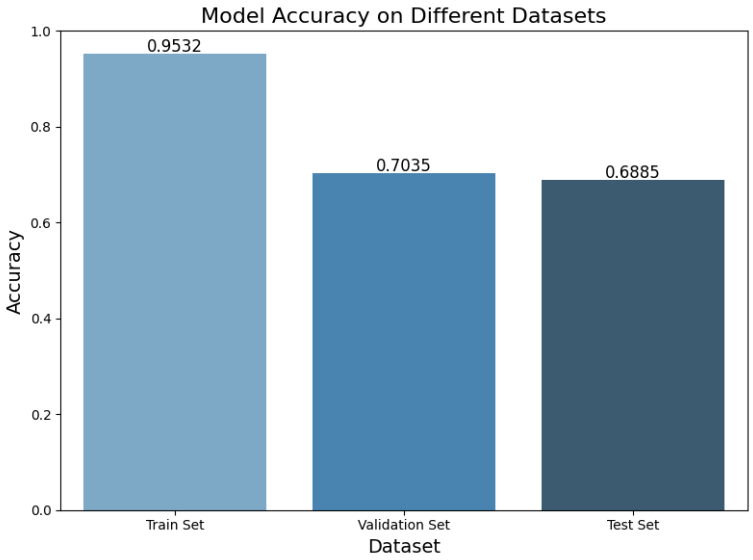

Figure 8: Accuracy scores for Train, Validation and Test set.

8 Usability & Scalabilty

A simple Dockerized API is provided for model inference, with the model stored in Google Drive and downloaded during the Docker build process. This setup offers basic scalability, as it allows for the deployment of multiple Docker containers to handle inference requests. However, a more scalable solution would involve implementing an API client that manages incoming requests and queues them in a database like Redis. Workers, utilizing frameworks like Celery, could then consume these tasks asynchronously, providing a more efficient and scalable architecture for handling large volumes of inference requests.

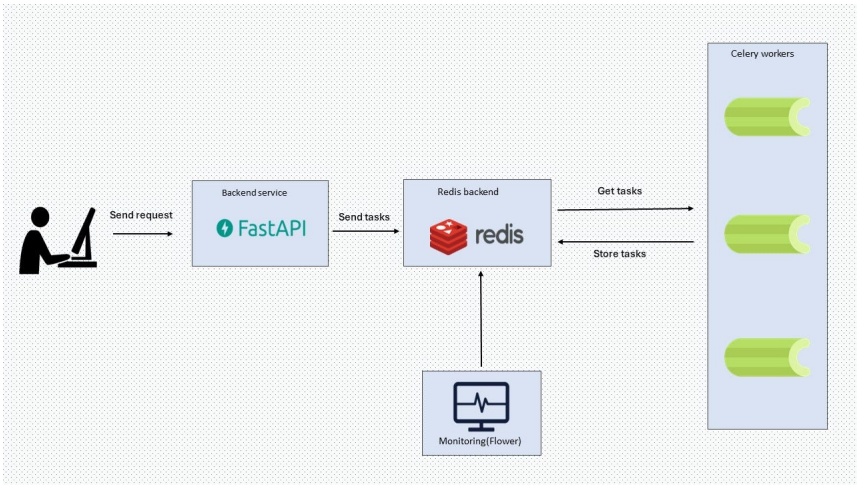

Figure 9: FastAPI, Celery, Redis architecture overview [5].

9 Conclusions

In this work, I have presented a proof-of-concept for a Siamese network model applied to the given task. The experiments demonstrated that, with proper fine-tuning of the model and hyperparameters, significant improvements

could be achieved in performance. However, due to time constraints, extensive optimization and fine-tuning were not performed in this study.

Several optimizations could further enhance the efficiency and effectiveness of the model. For example, batch preprocessing could reduce data preparation time, while utilizing multiprocessing techniques could accelerate the data loading and pair creation process, especially for large datasets.

The literature review also revealed that there are more sophisticated approaches [6] that can potentially yield better results, such as more advanced architectures or additional techniques for handling the data. However, due to the time limitations of this study, these more complex solutions were not explored in depth.

Overall, while the results achieved here serve as a solid foundation, further exploration and optimization could lead to more robust and accurate models in future work.

References

[1] F. Yesiler, C. Tralie, A. Correya *et al.*, "Da-TACOS: A dataset for cover song identification and understanding," in *Proc. of the 20th Int. Soc. for Music Information Retrieval Conf. (ISMIR)*, Delft, The Netherlands, 2019, pp. 327–334.

[2] A. A. Correya, R. Hennequin and M. Arcos, *Large-scale cover song detection in digital music libraries using metadata, lyrics and audio features*, 2018. arXiv: 1808.10351 [cs.IR]. [Online]. Available: https://arxiv.org/abs/1808.10351.

[3] J. Papa and A. Falcão, "A learning algorithm for the optimum-path forest classifier," vol. 5534, May 2009, pp. 195–204, ISBN: 978-3-642-02123-7. DOI: 10.1007/978-3-642-02124-4_20.

[4] M. Stamenovic, *Towards cover song detection with siamese convolutional neural networks*, 2020. arXiv: 2005.10294 [eess.AS]. [Online]. Available: https://arxiv.org/abs/2005.10294.

[5] Y. Chamrah, *Empowering applications with asynchronous magic: The celery, fastapi, docker, and flower*, Accessed: 2024-12-23, 2024. [Online]. Available: https://medium.com/@youssefchamrah/empowering-applications-with-asynchronous-magic-the-celery-fastapi-docker-and-flower-ac119efc2e04.

[6] F. Liu, D. Tuo, Y. Xu and X. Han, *Coverhunter: Cover song identification with refined attention and alignments*, 2023. arXiv: 2306.09025 [cs.SD]. [Online]. Available: https://arxiv.org/abs/2306.09025.