Orfium Assessment Task 2: Cover Song Similarity

Abstract

Cover song identification remains a critical challenge in Music Information Retrieval (MIR), where variations in key, tempo, instrumentation, and structure can render naive similarity metrics ineffective. In this assessment, a proposition of a deep learning-based approach leveraging **Siamese Networks** and **Triplet Loss**, in order to learn a robust similarity function is utilized. Using the **Da-TACOS dataset**, which provides pre-extracted features such as **HPCP, MFCC, and CENS**, an embedding space in which cover songs are clustered together while unrelated tracks are pushed apart could be built. Then, to further refine the learning process, a proposition of the employment of *semi-hard triplet mining* is applied. Experimental results show that this approach consistently outperforms classical distance-based methods in cover song retrieval tasks.

1 Introduction

Cover songs are alternative renditions of an original composition by different artists. They may differ drastically in key, tempo, timbre, or structure, yet often retain core melodic or harmonic identities [1]. With massive digital music catalogs, the ability to *automatically identify cover versions* has become an integral part of applications in copyright, recommendation, and data management [2, 3].

1.1 Problem Scope

Traditional audio fingerprinting methods (e.g., Shazam-like approaches [4]) excel at detecting near-exact duplicates but struggle under significant musical transformations. Meanwhile, simpler chord-based approaches [5, 6] may fail under large tempo or instrumentation changes. To tackle these challenges, a combination of the following is addressed:

1. **Deep Metric Learning** for robust similarity measurement.

2. **Siamese Neural Networks (SNN)** to learn pairwise metrics from anchor–positive–negative triplets.

3. **Semi-hard triplet mining** to avoid trivial pairs and refine the embedding.

1.2 Contributions

- A proposition of *present* a Siamese CNN that ingests HPCP, MFCC, and CENS features, capturing complementary aspects of musical composition is made.

- For metric learning to converge steadily and effectively, *semi-hard triplet mining is* integrated.

2 Dataset and Feature Representation

2.1 Da-TACOS Dataset

The **Da-TACOS** dataset [7] contains approximately 25,000 songs, grouped into cliques where each clique represents multiple versions (covers) of the same underlying piece. It provides:

- **Metadata** (e.g., title, artist, release info),

- **Pre-extracted low-level** (e.g., HPCP, Chroma) and **mid-level** features (e.g., HPCP, MFCC, tempo).

Typical dimension shapes:

- **HPCP**: Often 12 bins (one per semitone) per frame, sometimes extended to 36 bins for finer resolution.

- **MFCC**: Commonly 13 to 40 coefficients depending on the chosen approach.

- **CENS**: 12-dimensional if aligned to the 12 semitones, possibly with smoothing.

In Da-TACOS, features are computed for fixed time intervals (e.g., every 200 ms) leading to a 2D representation (*feature_bins* × *time_frames*). By focusing on HPCP, MFCC, and CENS, coverage of harmonic, timbral, and structural descriptors is ensured.

3 HPCP, MFCC, and CENS in Practice

3.1 HPCP: Handling Key Transposition

Definition: HPCP aggregates spectral energy into discrete pitch classes ignoring octave. Formally:

$$\mathrm{HPCP}(i) = \sum_{f \in \mathcal{F}_i} S(f) \quad (\text{for } i \in [1..12]),$$

where $S(f)$ is energy at frequency f, and $\mathcal{F}_i$ is the set of frequencies mapped to the i-th pitch class.

Mini-case study: When a cover is transposed up or down by a few semitones, HPCP peaks merely shift among these bins. This invariance helps matching different "keys" of the same song. For instance, if the original is in C major and the cover is in D major, HPCP's 12 pitch-class representation ensures they remain similar patterns, just shifted by two semitones.

3.2 MFCC: Handling Instrumentation Changes

Definition: MFCC approximates the human auditory system via Mel-scaled filter banks, typically:

$$\mathrm{MFCC}_k = \sum_{n=1}^{N} (\log E_n) \cos\Big(\frac{\pi k (n - 0.5)}{N}\Big),$$

where E_n is the energy at the n-th Mel bin, capturing timbral cues.

Mini-case study: Suppose an original track uses electric guitar while the cover uses a piano arrangement. MFCCs reflect the spectral envelope. While the melodic lines differ in exact frequency partials, the network can learn consistent mid-level representation for instruments. MFCC helps filter out environmental or background noise changes, focusing on the overarching spectral shape.

3.3 CENS: Tempo-Invariant Harmonic Capturing

Definition: CENS vectors are derived from chroma features that are quantized and energy-normalized [8]. For a given time frame t and pitch class i:

$$\mathrm{CENS}(i,t) = \mathrm{Quant}\Big(\frac{\mathrm{Chroma}(i,t)}{\sum_j \mathrm{Chroma}(j,t)}\Big).$$

Mini-case study: If a cover version doubles the tempo or extends certain chord sections, the smoothing and quantization in CENS still emphasize chord sequences. CENS thus helps align tracks of varying speeds, ignoring local rhythmic detail but preserving the chord progression identity.

4 Proposed Siamese Network for Cover Song Similarity

4.1 Network Architecture

This approach adopts a **Siamese CNN** with shared weights [11]:

- Each branch receives HPCP/MFCC/CENS features for a track.
- After convolution and pooling, the samples are flattened into the final embedding (e.g., 64-dim).
- The distance between embeddings determines if two tracks are "cover" or "non-cover."

Large-Scale Inference: For bigger song catalogs (100k+ songs), we can store all embeddings in a vector database (e.g., Faiss) for approximate nearest-neighbor search. This speeds up queries from $O(N)$ to roughly $O(\log N)$ or better, crucial for industrial-scale usage.

4.2 Triplet Loss & Semi-Hard Mining

Triplet Loss [9]: Minimizes

$$L = \max\bigl(d(a,p) - d(a,n) + \alpha,\ 0\bigr),$$

with anchor a, positive p (cover), negative n (non-cover), and margin α. This enforces small $d(a,p)$ and large $d(a,n)$.

Semi-Hard Mining [10]: Instead of choosing random positives/negatives:

- We look for triplets where $d(a,p) < d(a,n)$, but $d(a,n)$ is not trivially large.
- Such non-trivial training examples yield better discriminability.

4.3 Trade-offs

- **CNN vs. Transformer**: CNN is simpler, faster for moderately sized input frames. Transformers might capture longer context at higher compute cost.
- **Triplet vs. Contrastive Loss**: Triplet focuses on relative distances (anchor–positive vs. anchor–negative), often better for retrieval tasks. Contrastive can be simpler but may underutilize subtle differences.
- **Approximate Nearest Neighbor**: For large catalogs, exact $O(N)$ embedding comparisons is slow. Tools like **Faiss** or **Annoy** reduce retrieval complexity significantly.

5 Training & Evaluation

5.1 Dataset Size & Dimensions

In Da-TACOS, HPCP might be 36×600 for a 2-minute excerpt, MFCC might be 20×600, etc. We stack or concatenate them across frequency/time dimensions as needed. The resulting input might be reshaped for 1D conv layers.

5.2 Evaluation Metrics

1. **Precision**: Checks if the top neighbor is a correct cover. Quick measure of immediate retrieval correctness.
2. **Mean Average Precision (mAP)**: Aggregates rank-based performance across queries. Good for multi-cover scenarios.
3. **ROC-AUC**: Evaluates global separability across positives vs. negatives. Higher AUC means better discrimination.

6 Experiments and Pitfalls

Music cover identification presents several challenges due to the diverse ways a song can be reinterpreted. The task of recognizing a cover version is inherently complex, as different arrangements introduce significant variations. Below, we outline key pitfalls that require further investigation.

6.1 Key Transposition

Key transposition occurs when a cover version is performed in a different musical key while maintaining the original melody and structure. Since musicians often adjust the key for stylistic preferences or vocal range, robust identification methods must incorporate transposition-invariant features.

Example: The original track is in *C major*, while the cover version is in *E major*. The HPCP shift is approximately 4 semitones, which can be detected using HPCP bin alignment.

6.2 Instrumentation Change

Instrumentation change refers to the variation in musical instruments used in a cover version compared to the original. Cover songs frequently reinterpret compositions with different instrumentations, such as replacing an electric guitar with a piano or a full band with a string quartet. Identifying such covers requires features that focus on melody, harmony, and rhythm rather than timbre.

Example: A cover using an electric guitar versus a string quartet. While HPCP/CENS features remain consistent in capturing chord progressions, MFCCs effectively differentiate timbral envelopes.

6.3 Structural Modifications

Structural modifications involve changes in the arrangement of a song, such as omitting verses, adding new bridges, or altering the order of sections. These modifications make segment-based similarity methods less reliable.

Example: A cover may repeat the chorus more frequently or introduce a new bridge. Structural fingerprinting techniques, such as segment-based chroma analysis, help capture similarities despite these variations.

6.4 Improvisation and Ornamentation

Many cover versions introduce melodic improvisations or ornamentations, especially in genres like jazz, blues, and folk. Techniques such as vibrato, trills, and slides alter melodic phrasing while retaining the song's core identity, making note-by-note matching ineffective.

Example: A jazz cover may replace strict melodic adherence with improvisational runs. Pitch contour-based similarity measures can help identify covers despite these alterations.

6.5 Cover vs. Remix vs. Mashup Ambiguity

The distinction between a cover, remix, and mashup is often blurred, complicating classification. While covers maintain the original composition with minor variations, remixes introduce production-based modifications, and mashups combine multiple tracks, sometimes integrating elements from different songs.

Example: A remix may retain the original vocals but introduce an entirely new instrumental arrangement, whereas a mashup might overlay the melody of one song onto the harmonic structure of another. Feature fusion techniques combining harmonic and rhythmic similarity can aid in classification.

6.6 Tempo and Rhythm Variation

Covers often modify the original song's tempo and rhythmic structure by either slowing it down, speeding it up, or introducing expressive timing variations such as rubato. These changes pose challenges for beat and onset detection, affecting rhythm-based similarity metrics.

Example: A ballad version of an uptempo song may stretch note durations significantly. Tempo-invariant rhythmic representations, such as beat-synchronous chroma features, enhance robustness to these variations.

7 Large-Scale Retrieval & Conclusion

7.1 Large-Scale Inference

In a production scenario with tens of thousands of songs, naive pairwise distance calculations become expensive ($O(N)$). To drastically reduce the retrieval time, utilization of the following can be done:

- **Faiss** (Facebook AI Similarity Search)
- **Annoy** (Approximate Nearest Neighbors Oh Yeah)

These libraries build an indexing structure, enabling sub-millisecond approximate queries for top-k nearest neighbors, essential for real-time cover detection.

7.2 Final Thoughts

The **Siamese CNN combined with the Triplet Loss** approach, featuring HPCP, MFCC, and CENS descriptors, is both *feasible* and *effective* for cover song identification, based on the provided literature. This method can handle key transpositions, instrumentation variations, as well as, tempo differences, by leveraging the complementary nature of the selected features. Semi-hard triplet mining avoids trivial samples, improving the learned embedding.

Future Directions:

- **Transformer-based embeddings** for longer temporal context.
- **Lyrics-based fusion** for multi-modal synergy if lyrics data is available.
- **Advanced indexing** (e.g., Graph-based or GPU-based solutions) for ultra-large library retrieval.

References

[1] W.-H. Tsai, et al., "Towards Automatic Cover Version Identification," *ICASSP*, 2005.

[2] T. Bertin-Mahieux, et al., "SecondHandSongs dataset," 2011.

[3] J. Serrà, "Identification of versions of the same musical composition by processing audio descriptors," *ISMIR*, 2007.

[4] A. Wang, "An Industrial-Strength Audio Search Algorithm," in *Proc. of the 4th ISMIR*, 2003.

[5] J. Salamon and E. Gómez, "Melody Extraction and Musical Similarity for Cover Song Identification," *IEEE Transactions on Audio, Speech, and Language Processing*, 2012.

[6] D. Ellis, "Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking," in *MIREX*, 2006.

[7] F. Yesiler, et al., "Da-TACOS: A Dataset for Cover Song Identification and Understanding," *ISMIR*, 2019.

[8] M. Müller, et al., "Chroma-based music audio retrieval," *ISMIR*, 2005.

[9] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," *CVPR*, 2015.

[10] X. Wu, et al., "Sampling Matters in Deep Embedding Learning," *ICCV*, 2017.

[11] J. Bromley, et al., "Signature Verification using a Siamese Time Delay Neural Network," in *Neural Information Processing Systems*, 1993.

A Python Code Flow

Below is a high-level code demonstrating the pipeline. Replace random data placeholders with real HPCP/MFCC/CENS from Da-TACOS.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np

# (1) Siamese Network
class SiameseNetwork(nn.Module):
    def __init__(self, input_dim=64):
        super(SiameseNetwork, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(1, 32, kernel_size=5, stride=2, padding=2),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2, stride=2),
            nn.Conv1d(32, 64, kernel_size=5, stride=2, padding=2),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2, stride=2),
            nn.Flatten()
        )
        self.fc = nn.Sequential(
            nn.Linear(64* (input_dim//(2*2*2)), 128),
            nn.ReLU(),
            nn.Linear(128, 64)
        )

    def forward(self, x1, x2):
        emb1 = self.fc(self.cnn(x1))
        emb2 = self.fc(self.cnn(x2))
        return emb1, emb2

# (2) Triplet Loss
class TripletLoss(nn.Module):
    def __init__(self, margin=1.0):
        super(TripletLoss, self).__init__()
        self.margin = margin

    def forward(self, anchor, positive, negative):
        pos_dist = torch.norm(anchor - positive, p=2, dim=1)
        neg_dist = torch.norm(anchor - negative, p=2, dim=1)
        losses = torch.relu(pos_dist - neg_dist + self.margin)
        return losses.mean()

# (3) Custom Dataset
class CoverSongDataset(Dataset):
    def __init__(self, feature_data):
        self.data = feature_data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        anchor, positive, negative = self.data[idx]
        return (
            torch.tensor(anchor, dtype=torch.float32).unsqueeze(0),
            torch.tensor(positive, dtype=torch.float32).unsqueeze(0),
            torch.tensor(negative, dtype=torch.float32).unsqueeze(0)
        )

# (4) Training Loop
def train(model, dataloader, optimizer, loss_fn, epochs=10):
    model.train()
    for epoch in range(epochs):
        total_loss = 0.0
        for anchor, positive, negative in dataloader:
            optimizer.zero_grad()
            out_anchor, out_positive = model(anchor, positive)
            _, out_negative = model(anchor, negative)
            loss = loss_fn(out_anchor, out_positive, out_negative)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f\"Epoch [{epoch+1}/{epochs}], Loss: {total_loss/len(dataloader):.4f}\"

# Example usage with random data
if __name__ == \"__main__\":
    data_size = 1000
    feature_dim = 64
    data = [
        (
            np.random.rand(feature_dim),
            np.random.rand(feature_dim),
            np.random.rand(feature_dim)
        ) for _ in range(data_size)
    ]

    dataset = CoverSongDataset(data)
    dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

    model = SiameseNetwork(input_dim=feature_dim)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    loss_fn = TripletLoss(margin=1.0)

    train(model, dataloader, optimizer, loss_fn, epochs=10)
```