Cover song similarity

Dimitris Loupas

August 2025

1 Introduction

In this report, we explore *cover song similarity* with neural networks—a classic task in Music Information Retrieval. We work on the Da-TACOS dataset and compare two models: (i) a classical Convolutional Neural Network, which is a *Siamese* 1D CNN that learns embeddings and compares pairs directly; and (ii) a hybrid variant that keeps the same backbone but inserts a small quantum layer on top of the embedding to see if it gains any extra performance. The goal isn't to exhaust every design choice—given time constraints, we focus on a simple comparison. We first sketch the dataset and features, then describe the two architectures and training setup, and finally report results.

2 Dataset

We use *Da-TACOS*.[1] [2] (cover–song dataset), which follows the SecondHandSongs setup: each "work" is a cover clique and each "performance" is a specific recording. Filenames are the performance IDs (`PID`, e.g., `P_22`) and the class label is the work ID (`WID`, e.g., `W_14`). The corpus comes in two subsets: a **benchmark** subset (15k performances) for running fair comparisons, and a **cover-analysis** subset (10k) for investigate how versions relate. No audio here—only pre-extracted features and metadata (which is fine for our experiments).

[1] `https://github.com/MTG/da-tacos`
[2] `https://mtg.github.io/da-tacos/`

- **Organization.** Data are structured by WIDs (works) and their PIDs (performances). This makes it straightforward to retrieve all versions of the same work and to form positive/negative pairs for training.

- **Features (HDF5).** Features were extracted from 44.1 kHz MP3 audio and released as HDF5 files in two layouts: (i) *single-files per performance*, where each H5 contains multiple entries for one song; and (ii) *per-feature folders* that store only one feature per song. Common keys include `hpcp`, `crema` (CREMA-PCP), `chroma_cens`, `mfcc_htk`, `key_extractor` (`key`, `scale`, `strength`), `madmom_features` (`novfn`, `snovfn`, `onsets`, `tempos`), `tags`, plus `label` and `track_id`.

- **Metadata (JSON).** Metadata live under `da-tacos_metadata` as JSON dictionaries keyed by `WID`. Each WID entry lists its PIDs with fields such as `perf_title`, `perf_artist`, `work_title`, `work_artist`, `release_year`, SecondHandSongs `perf_id`/`work_id`, and an `instrumental` flag. Where available, records are augmented with MusicBrainz matches (e.g., artist MBIDs, per-performance MBIDs with `length`, and genre/style tags). Coverage of these enrichments is partial.

- **Intended use.** The benchmark subset is designed for quantitative evaluation of cover identification systems; the cover-analysis subset supports qualitative/quantitative studies of musical variation across versions.

In our analysis, we stick to the **benchmark** part of Da-TACOS. It has about **15,000** performances; where **1,000** work IDs that each have **13** covers, so roughly **13,000** are cover versions and the remaining **2,000** are non-covers. We focus on two features: **HPCP** (Harmonic Pitch Class Profile; a 12-D pitch-class energy vector per frame that tracks harmonic content) and **CREMA** (a chroma/PCP variant designed to be a bit more robust to timbre and tuning drift). The two feature distributions we use are shown in the plots below.

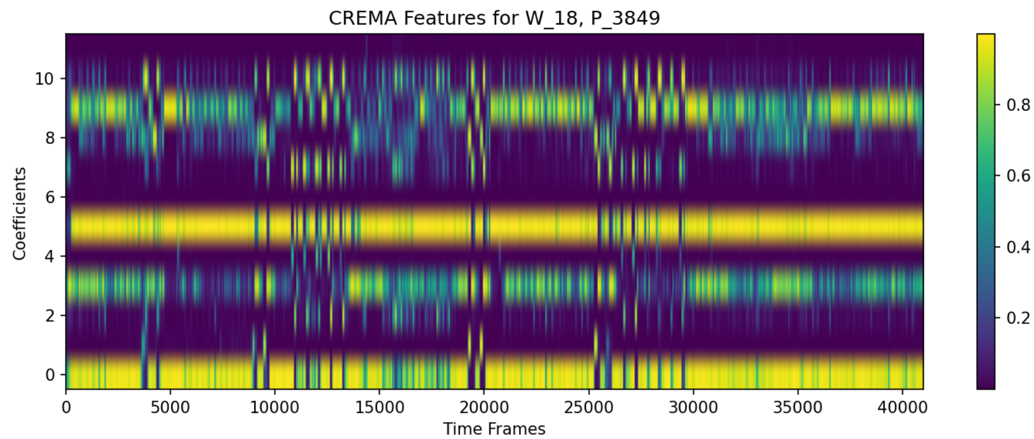

Figure 1: CREMA feature

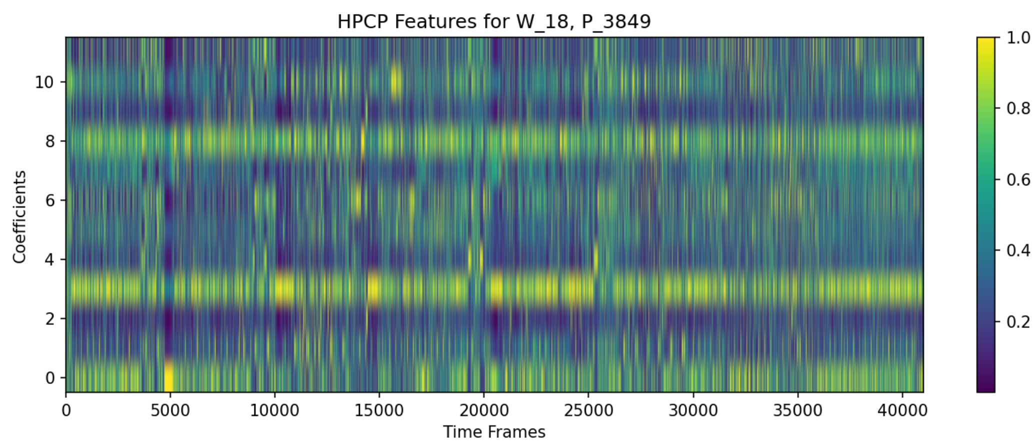

Figure 2: HPCP feature

Although raw values lie in $[0, 1]$, we also *normalize each frame to unit L2 norm* so that every 12-D vector has length 1. The histogram of sequence lengths is also shown below; the mean duration is about **18,124.5 ms**.

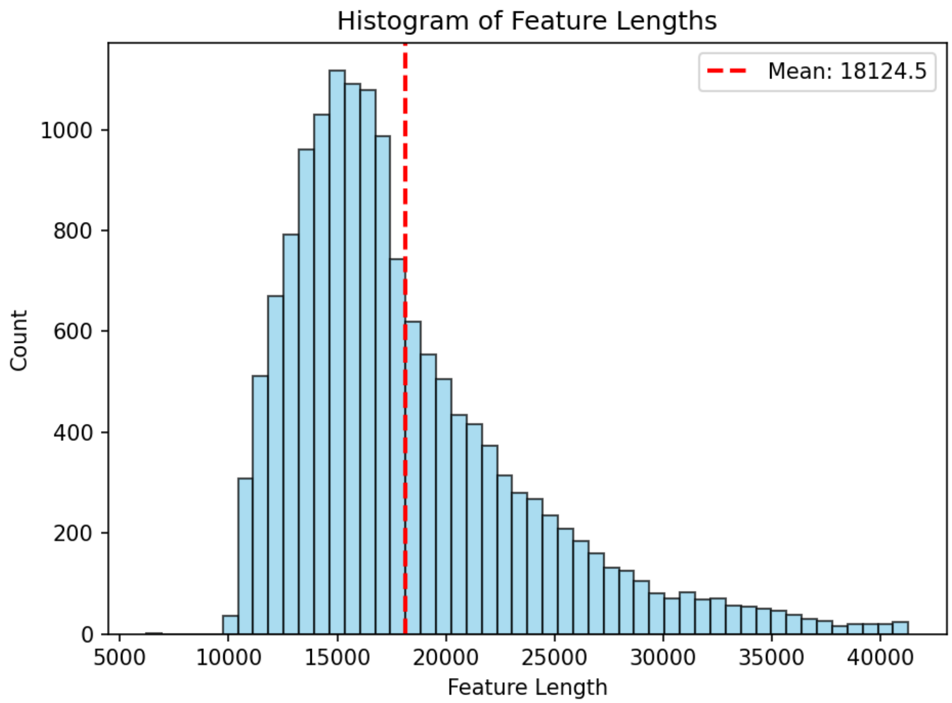

Figure 3: Histogram of feature length

Due to computational constraints, we **downsample each sequence to a total of 2,000 frames**. If a sequence is still longer after downsampling, we *truncate* it to 2,000; if it is shorter, we *right–pad with zeros* so that all inputs have the same length.

3 Model Architectures

We keep things simple and compare two variants of the same Siamese idea: (i) a **1D CNN with residual blocks** that turns a time–feature sequence into a compact embedding, and (ii) the same backbone but with a small **quantum layer** dropped after the embedding layer. The overall logic follows the Siamese setup in [1] (two networks with shared weights, compare their outputs, train end-to-end).

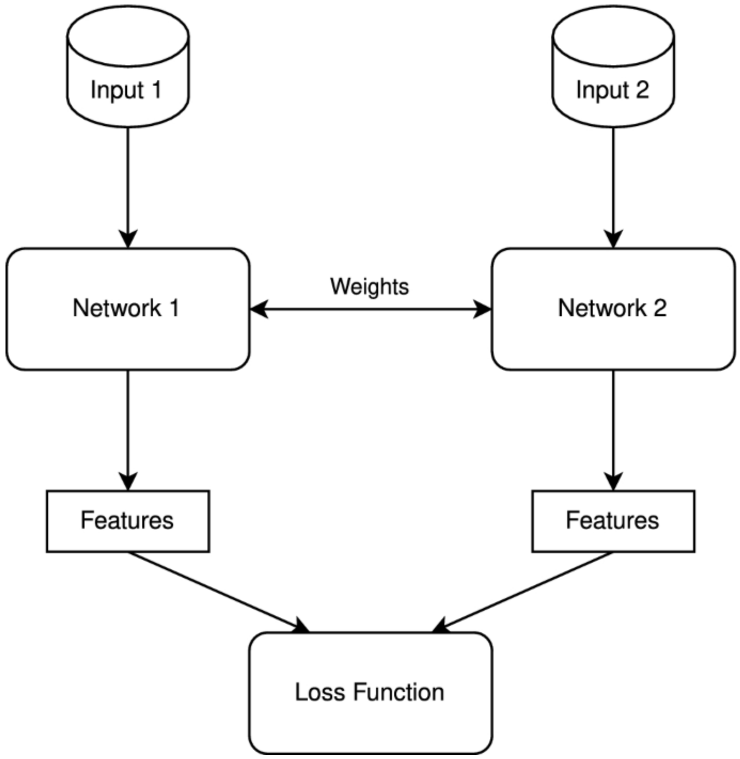

Figure 4: Siamese architecture. Source [2]

Comparator (same for both models). Each branch outputs an embedding $\mathbf{v}_a, \mathbf{v}_b \in R^D$ (we use D=128). We score the pair with a weighted squared distance pushed through a sigmoid:

$$p(x_a, x_b) \;=\; \sigma\left(\sum_{j=1}^{D} \alpha_j \,(v_a^{(j)} - v_b^{(j)})^2\right),$$

where $\sigma(\cdot)$ is the logistic function and α_j are learnable weights.

Loss (binary cross-entropy). We train with BCE on the pair label $y(x_a, x_b)$, where we set y=0 **for cover (similar)** and y=1 **for non-cover (dissimilar)**:

$$\mathcal{L}(x_a, x_b) \;=\; -\Big[\,(1 - y(x_a, x_b))\, \log p(x_a, x_b) \;+\; y(x_a, x_b)\, \log(1 - p(x_a, x_b))\Big].$$

Threshold. We use a fixed decision threshold of 0.5 on the predicted similarity probability $p(x_a, x_b)$. Pairs with $p \geq 0.5$ are classified as *non-cover* (label 0), and pairs with $p < 0.5$ as *cover* (label 1). All accuracies reported use this rule, while ROC curves are threshold–free and summarize performance across all possible thresholds.

Networks.

- **Residual 1D CNN.** Four residual blocks (Conv1d+BN+ReLU with skips) with time pooling between blocks, then global average/max pooling and a small MLP to get $\mathbf{v} \in R^{128}$.
- **Hybrid (quantum) variant.** Same CNN up to the embedding; then we project to the required size, run a small variational quantum circuit (AmplitudeEmbedding on q qubits + entanglers, readout as $E[Z]$ per qubit), map back to 128-D to get the embedding. Everything is still trained end-to-end using pennylane.

The two architectures are summarized in the next two tables

Table 1: Residual 1D CNN (FeatureExtractor) and Siamese head. Input is a time–feature matrix $[B, T, F]$ (e.g., F=12 for HPCP or F=24 for HPCP+CREMA).

#	Layer	Parameters / Notes
1	Input	$[B, T, F]$
2	Transpose	$[B, F, T]$ (channels-first for Conv1d)
3	**ResBlock1D**	$F \to 32$, two Conv1d(k=3, s=1, p=1) + BN + ReLU + Dropout(0.2);
4	MaxPool1d	kernel= 4, stride= 4 (time downsample ×4
5	**ResBlock1D**	$32 \to 64$, same as above
6	MaxPool1d	kernel= 4, stride= 4
7	**ResBlock1D**	$64 \to 128$, same as above
8	MaxPool1d	kernel= 4, stride= 4
9	**ResBlock1D**	$128 \to 256$, same as above
10	MaxPool1d	kernel= 4, stride= 4
11	Global AvgPool1d	output $[B, 256, 1] \Rightarrow [B, 256]$
12	Global MaxPool1d	output $[B, 256, 1] \Rightarrow [B, 256]$
13	Concatenate	$[B, 512]$ (avg $\|$ max)
14	Dense	$512 \to 512$, ReLU
15	Dropout	$p = 0.3$
16	Dense	$512 \to 128$ (embedding)
17	L2 Normalization	unit-norm embedding $[B, 128]$
18	**Siamese sharing**	two branches share weights of layers 1–17
19	Squared difference	$(\mathbf{v}_1 - \mathbf{v}_2)^2 \in R^{128}$
20	Dense (head)	$128 \to 1$, Sigmoid (pair similarity score)

Table 2: Residual 1D CNN + **Quantum** block (Siamese). Same backbone as Table 1, with a quantum layer inserted on top of the 128-D embedding.

#	Layer	Parameters / Notes
1–17	Residual 1D CNN backbone	As in Table 1; produces $[B, 128]$ embedding (L2-normalized
18	(Optional) Pre-projection	If $2^q \neq 128$: Linear $128 \to 2^q$, ReLU.
19	AmplitudeEmbedding	Input length 2^q over q wires; normalized amplitudes.
20	Entangling circuit	`BasicEntanglerLayers` with L layers on q wires.
21	Quantum readout	Expectation values $\{E[Z_i]\}_{i=1}^{q} \Rightarrow [B, q]$.
22	Post-projection	Linear $q \to 128$, ReLU (map back to 128-D).
23	L2 Normalization	Unit-norm embedding $[B, 128]$ (after quantum block).
24	**Siamese sharing**	Two branches share all weights (backbone + quantum).
25	Squared difference	$(\mathbf{v}_1 - \mathbf{v}_2)^2 \in R^{128}$.
26	Dense (head)	$128 \to 1$, Sigmoid (pair similarity score).

Notes. (i) If you set q=7 (so 2^q=128), step 18 is skipped because the CNN output already matches the amplitude size. (ii) If you set q=6 (2^q=64), step 18 uses Linear 128→64 before the quantum layer, and step 22 maps q=6 back to 128 with Linear 6→128. (iii) The quantum node returns q expectation values (E$[Z]$ per wire).

4 Results

In this section, we present the results of our analysis. We consider a dataset of 6000 total song pairs, where 50% are similar (song covers) and the remaining 50% are not. We split the dataset as follows:

Table 3: Splitting of the total dataset in training, validation and test sets

Total dataset	**Training set**	**Validation set**	**Test set**
5000	3000	1000	1000

and used the following hyperparameters.

Table 4: Hyperparameters

epochs	**learning rate**	**optimizer**	**loss**
20	0.0005	Adam	Binary cross entropy

CNN

The training and validation curves for the CNN are shown in Figs. 5–6. The **train loss** steadily decreases, while the **validation loss** decreases at first and then oscillates around a higher plateau. This gap points to mild *overfitting* after the first few epochs.

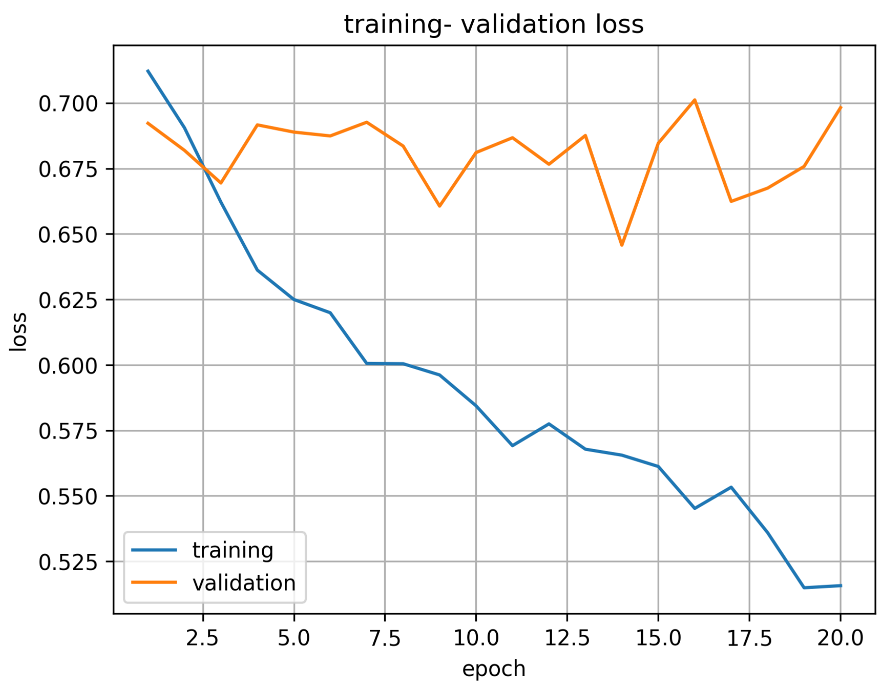

Figure 5: Training and validation loss (CNN).

Validation accuracy rises early but then fluctuates, whereas train accuracy keeps improving. The training accuracy reaches about ~ 0.74 while the best validation accuracy is around ~ 0.63. This suggests the model is learning useful patterns but overfits a bit on the training data.

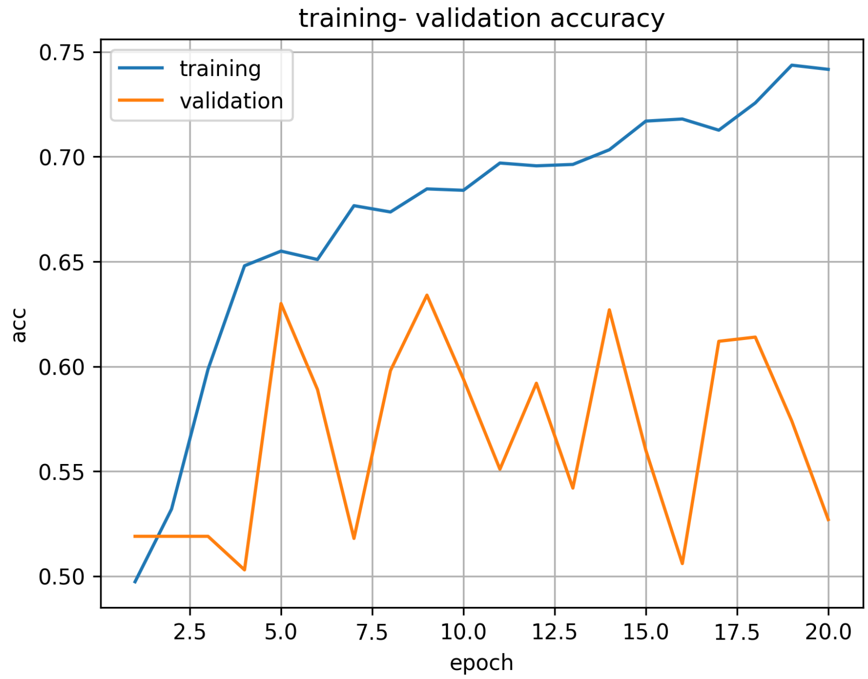

Figure 6: Training and validation accuracy (CNN).

The ROC curve confirms there is signal: AUC ≈ 0.70, well above random. **All ROC curves** are computed on the *test set* using the checkpoint with the best validation accuracy.

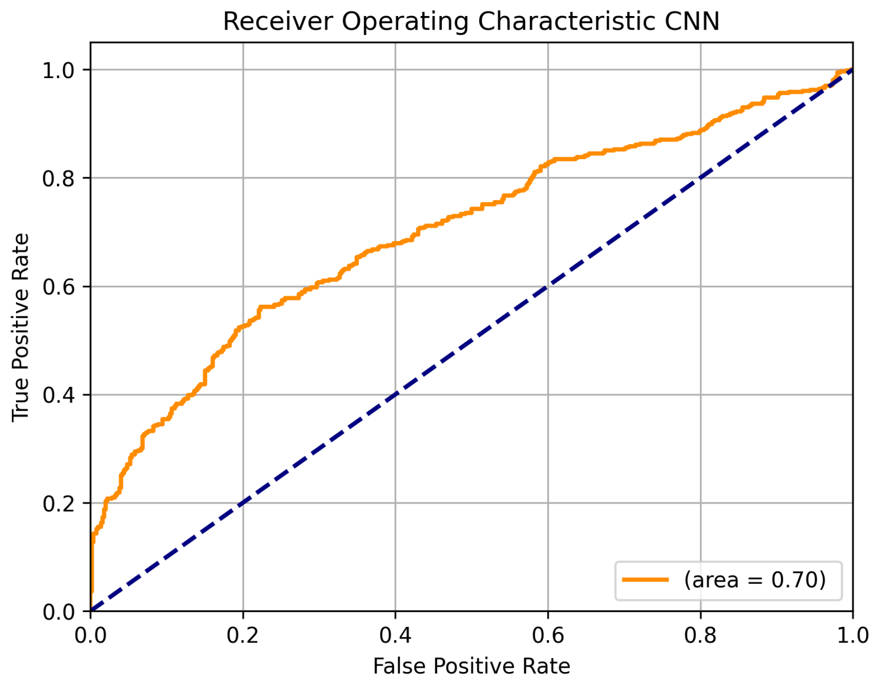

Figure 7: ROC curve (CNN), evaluated on the test set with the best validation checkpoint.

Comments (CNN).

- Clear train/val gap $\Rightarrow$ use **early stopping** near the first validation-loss minimum and/or a small LR scheduler.
- Test accuracy for this model is **0.62**.

CNN with Quantum Layer

With the quantum layer inserted, the overall behavior is similar: train loss decreases smoothly; validation loss drops early and then plateaus with small oscillations.

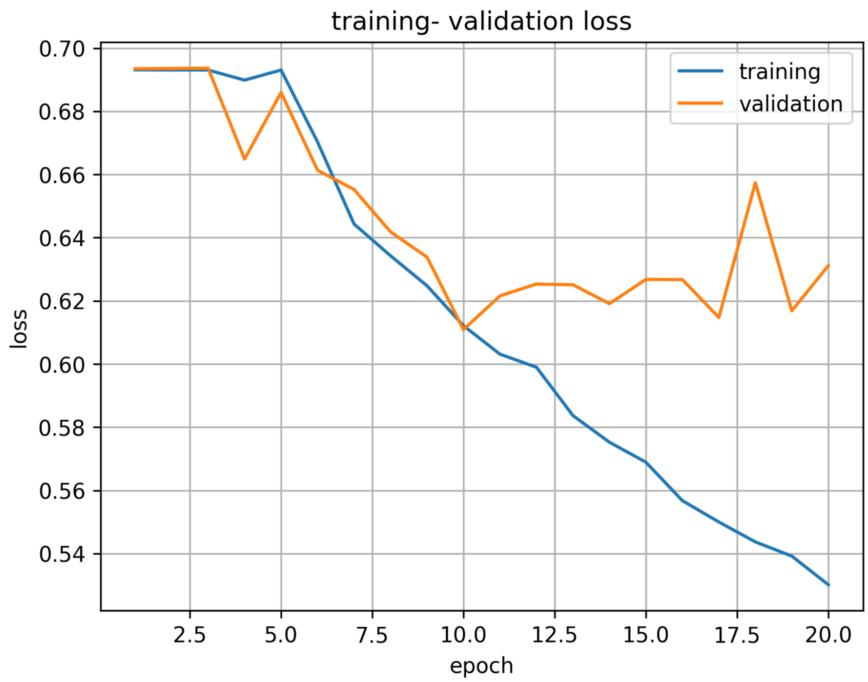

Figure 8: Training and validation loss (CNN + quantum).

Train accuracy also reaches ~ 0.74, while the best validation accuracy is about ~ 0.68.

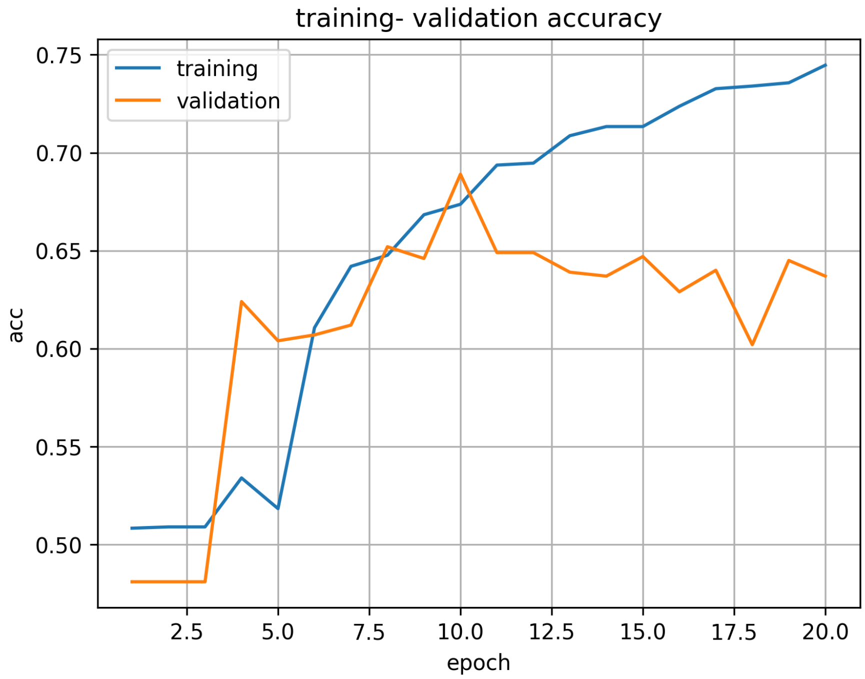

Figure 9: Training and validation accuracy (CNN + quantum).

The ROC curve shows a small but consistent improvement: AUC ≈ 0.71 (vs. ~ 0.70 for the plain CNN). As above, ROC is computed on the *test set* using the best validation checkpoint.

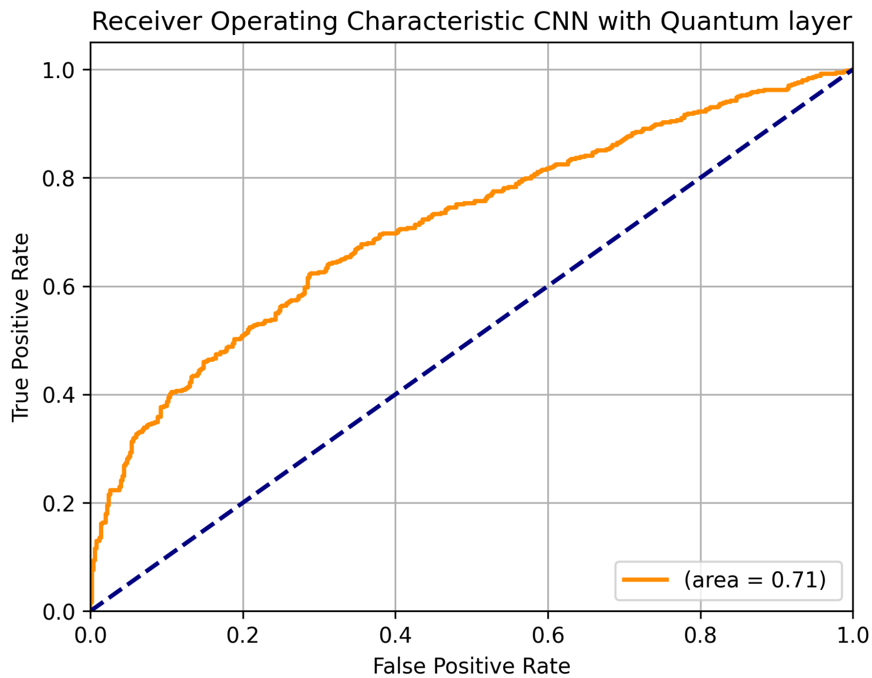

Figure 10: ROC curve (CNN + quantum), evaluated on the test set with the best validation checkpoint.

Comments (CNN + quantum).

- The AUC increase suggests a slight *ranking* gain; ideally verify with multiple random seeds for robustness.
- Test accuracy for this model is **0.65**.

5 Conclusion

We trained two Siamese models on the Da-TACOS benchmark subset, using BCE on a learned squared–difference head and evaluating with the checkpoint that achieved the best validation accuracy. The plain 1D-CNN reaches a **test accuracy of 0.62** with **AUC** ≈ 0.70; adding a small *quantum* block gives **0.65** test accuracy and **AUC** ≈ 0.71.In short, the quantum variant yields a small improvement in AUC and a modest gain in accuracy.

Learning curves look similar for both models: training loss/accuracy improve steadily, while validation flattens early and oscillates, indicating overfitting. With the quantum layer, the validation curves oscillate much less and are smoother, which makes the "when to stop" signal cleaner.

Limitations. We downsample each song to **2000 frames** for speed, which can discard useful temporal context. We also used **5,000 training pairs**—compact and reproducible, but not huge—so some variance across runs is expected. Finally, we reported accuracy with a fixed threshold of 0.5; calibrating the threshold on the validation set can yield higher accuracy.

Next steps.

- **Threshold calibration:** tune the decision threshold on the validation set (or use temperature/Platt scaling).
- **Regularization:** slightly stronger dropout/weight decay; consider GroupNorm if batch sizes are small.
- **Robustness (seeds):** repeat training with multiple random *seeds* (different initializations/shuffles) and report mean±std; add confidence intervals for AUC.
- **Quantum block:** repeat across seeds to assess whether the small gain is consistent and worth the compute cost.
- **Architecture:** Explore more architectures.

Overall, the baseline already learns a useful embedding ($AUC \approx 0.70$). The hybrid model improves class separability—positives are closer and negatives farther—yielding a small AUC gain and a modest accuracy improvement. More experiments are needed to establish significance and validate the result.

References

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

[2] N. Serrano and A. Bellogín, “Siamese neural networks in recommendation,” *Neural Computing and Applications*, vol. 35, pp. 13 941–13 953, 2023. [Online]. Available: https://doi.org/10.1007/s00521-023-08610-0.