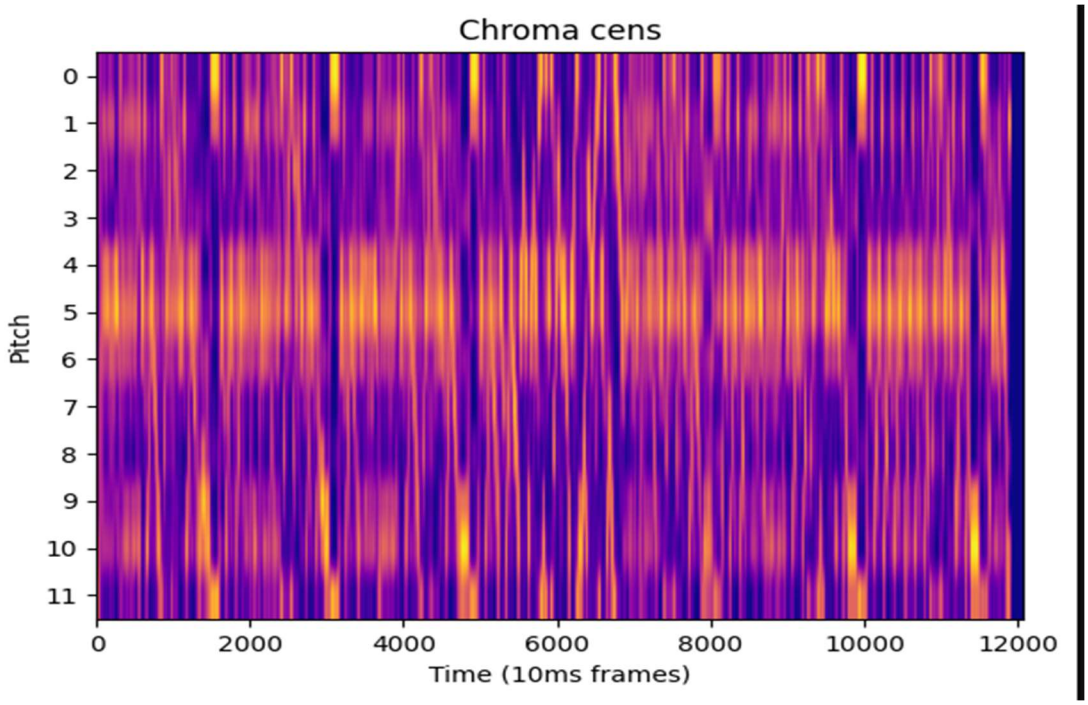
B. Hiring Assessment: Cover Song Similarity

The dataset contains pre-extracted features for songs (originals and covers) which can be used for cover song identification. We use the cover analysis subset, where data are organized in folders such that each folder contains two performances that refer to the same song (could be the original and a cover or two covers of the same original song).

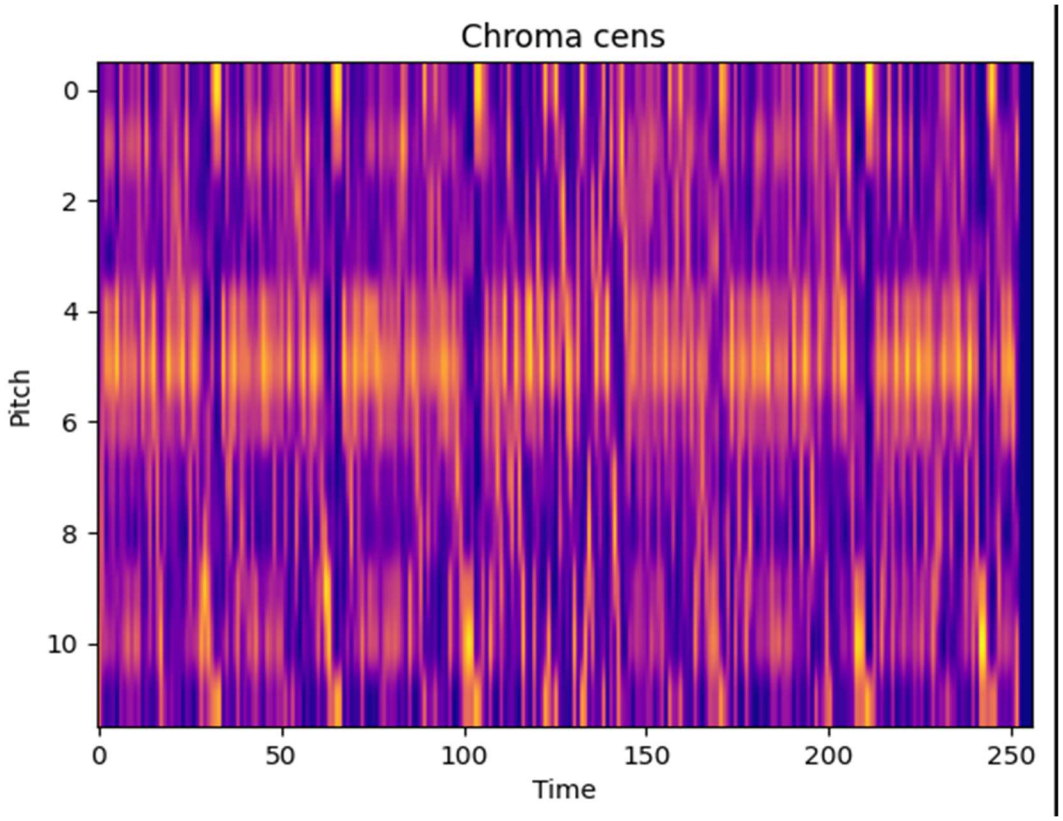
At first, we explore the data to get an idea of the features and how/which of them we can use for designing a model that measures similarity,

Features

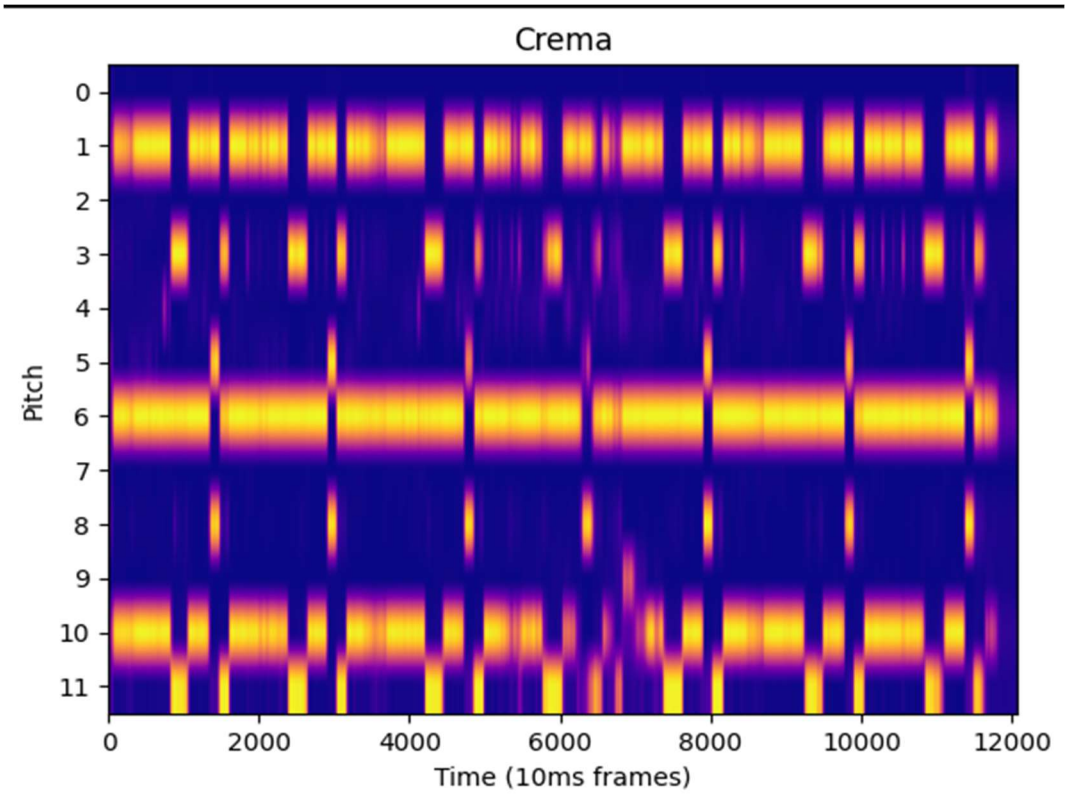
1. The first feature is called chroma_cens. It is a numpy array with a fixed number of 12 columns for all samples and different number of rows for each sample. Apparently, the chroma_cens is a pretty common feature in music data analysis with each column corresponding to one different pitch (i.e. A, A#, B, B#, etc). The rows correspond to different timeframes (most probably timeframes of 10ms) and since each song has a different duration, the number of rows is not fixed. These data seem to contain important information when displayed as images (below is an example of the chroma_cens feature) so we decide to use them in this form (we rotate the array by 90 degrees so the time is on the x-axis). Since they are of very high dimensionality (in the example displayed it is 12x12067), we resize them so that they are more suitable for machine learning algorithms, without losing critical information.



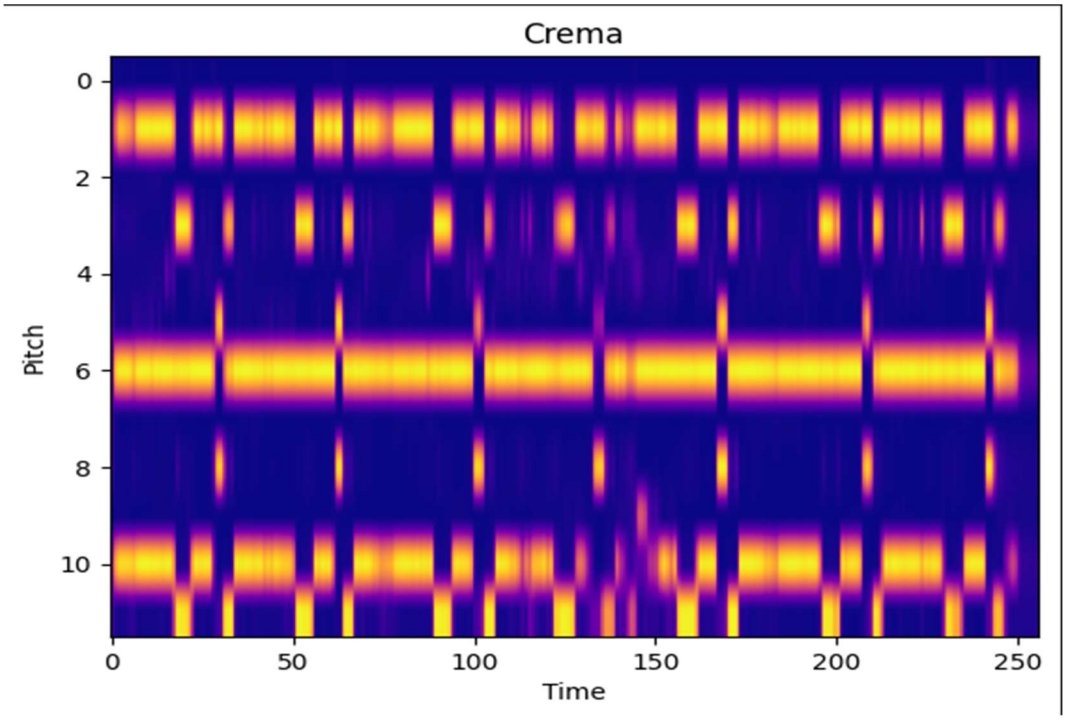
The final image is of size 12x256 as depicted below. It is clear that even though the dimensionality is reduced, the important part of the information is still present and can be useful for measuring the songs' similarity.



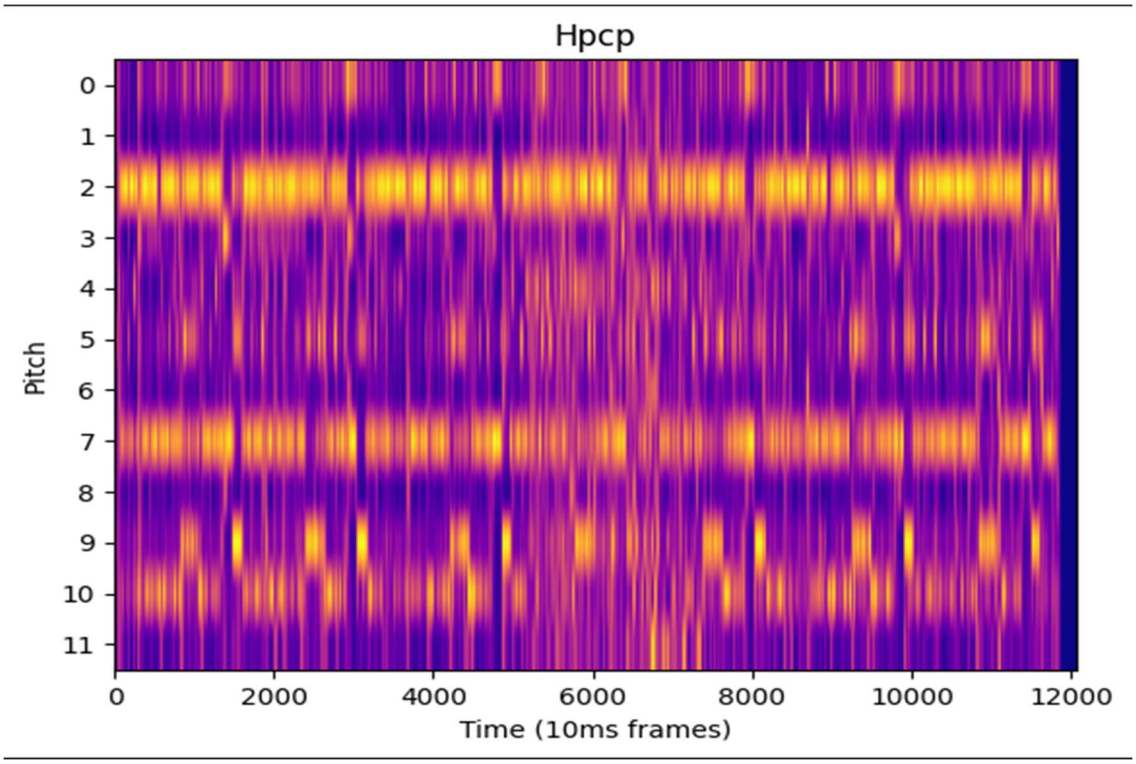
2. The second feature is called crema and is very similar to chroma_cens in terms of format. It is also an image with 12 rows corresponding to the 12 pitches and rows corresponding to timeframes. It contains probably information more related to the rhythm of the song. An example of the crema feature is shown below.

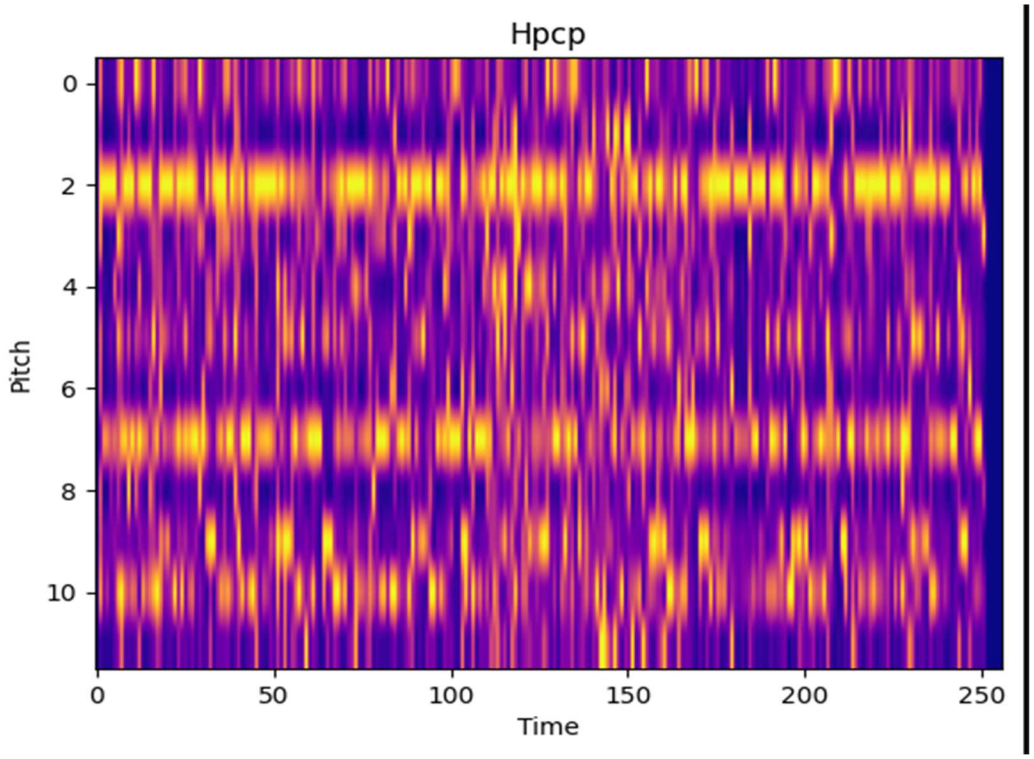


Again, we resize the image in 12x256 in order to preserve important patterns and reduce its dimensionality



3. Same logic for the third feature, hpcp, which is also an image. Below is an example before and after resizing.



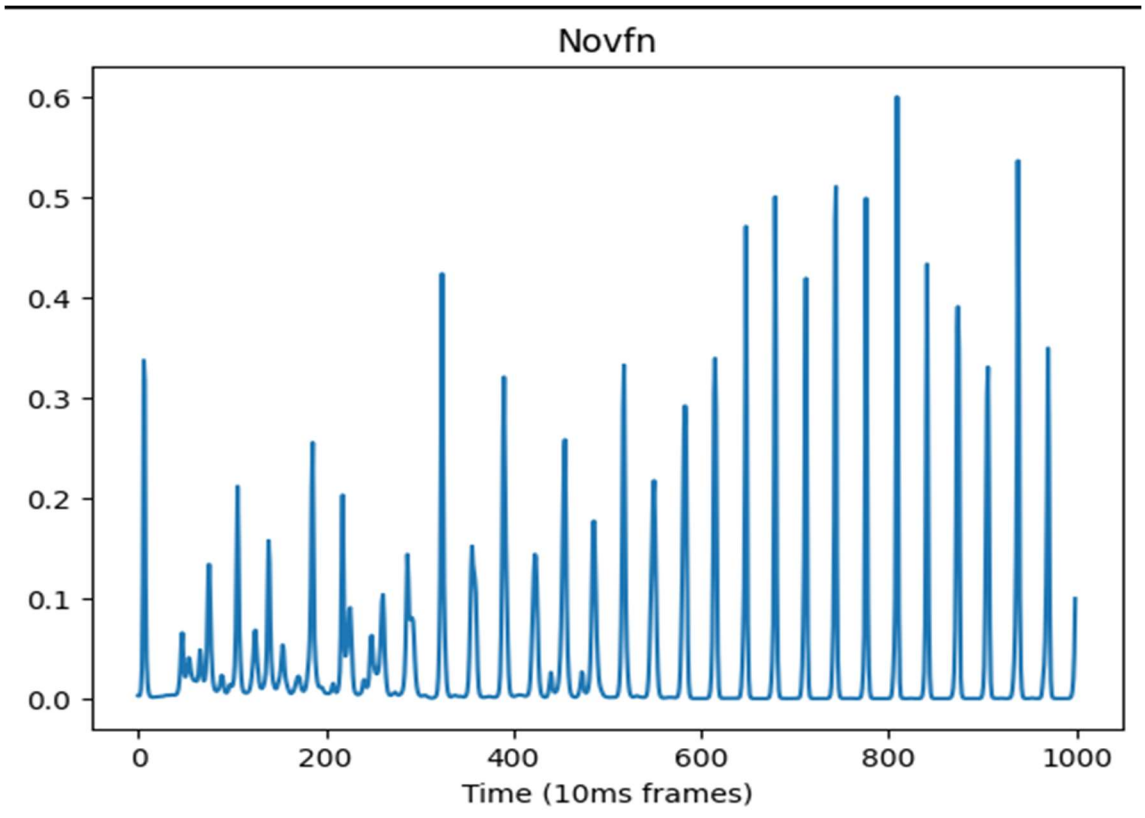


4. Next feature is a dictionary called key_extractor. It contains three values:
    - key (e.g. F#)
    - scale (e.g. minor)
    - strength (float e.g. 0.9458)

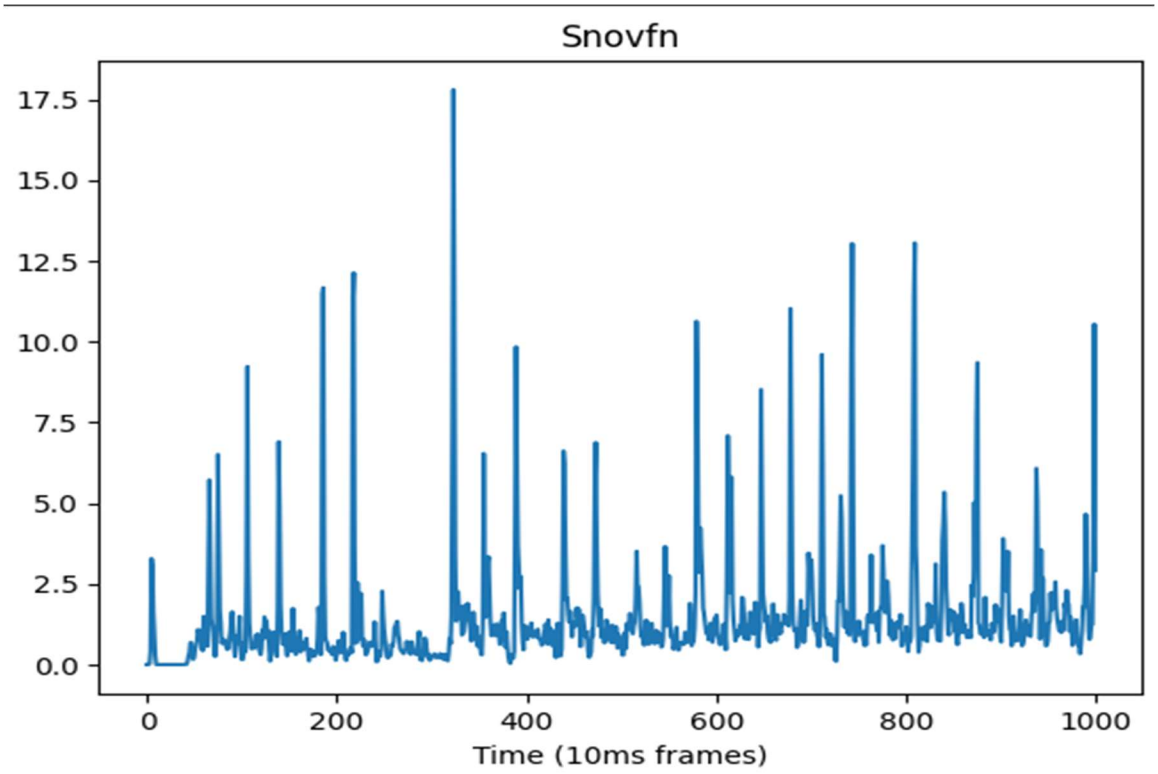
    Strength is possibly a probability on how certain we are about the key. Since there is only one option we are not going to use that. We concatenate key and scale in one string (e.g. F#minor) which we are going to use for the model.

5. Following, is a dictionary called madmom with 4 numpy arrays. These seem to be related to the rhythm of the song.

    - onsets: this array contains information on the timesteps that a new note begins (e.g. [6, 42, 76, 111, …])
    - novfn: it is not very clear what this feature represents but its peaks are in similar timeframes with the onsets. Its shape is also (12067,) meaning the x-axis represents clearly timeframes, we just plot the first 1000 so the image is more clear

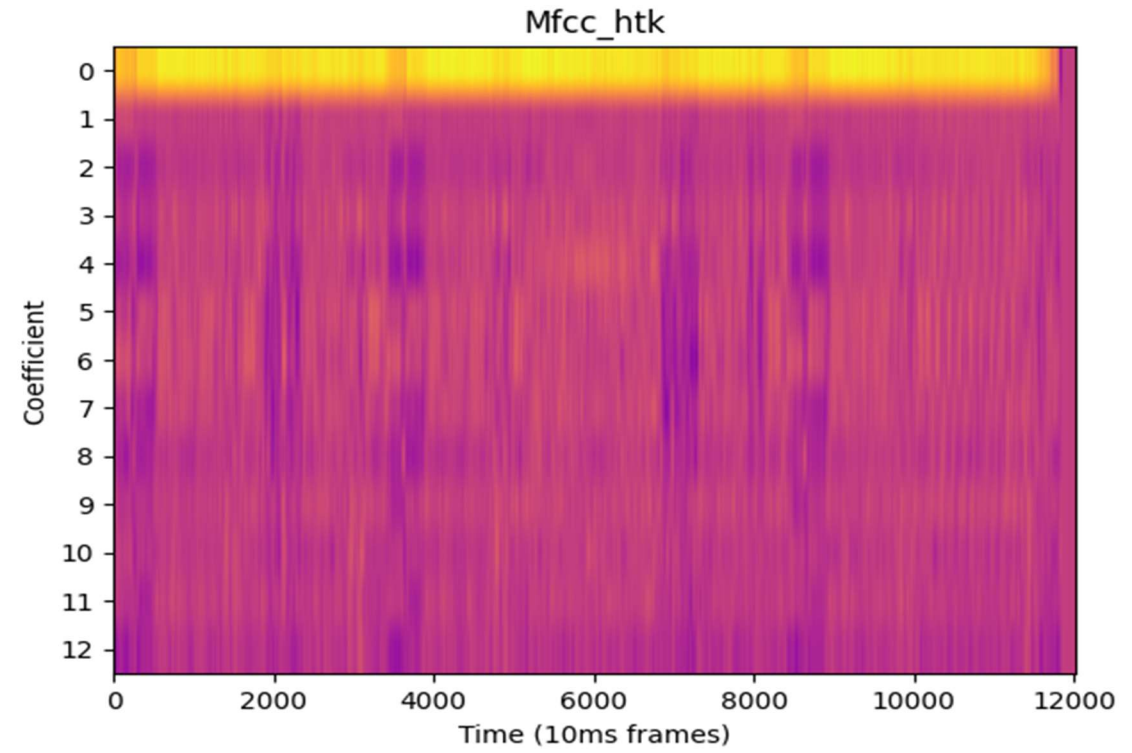


- snovfn: probably an enhanced version of novfn.

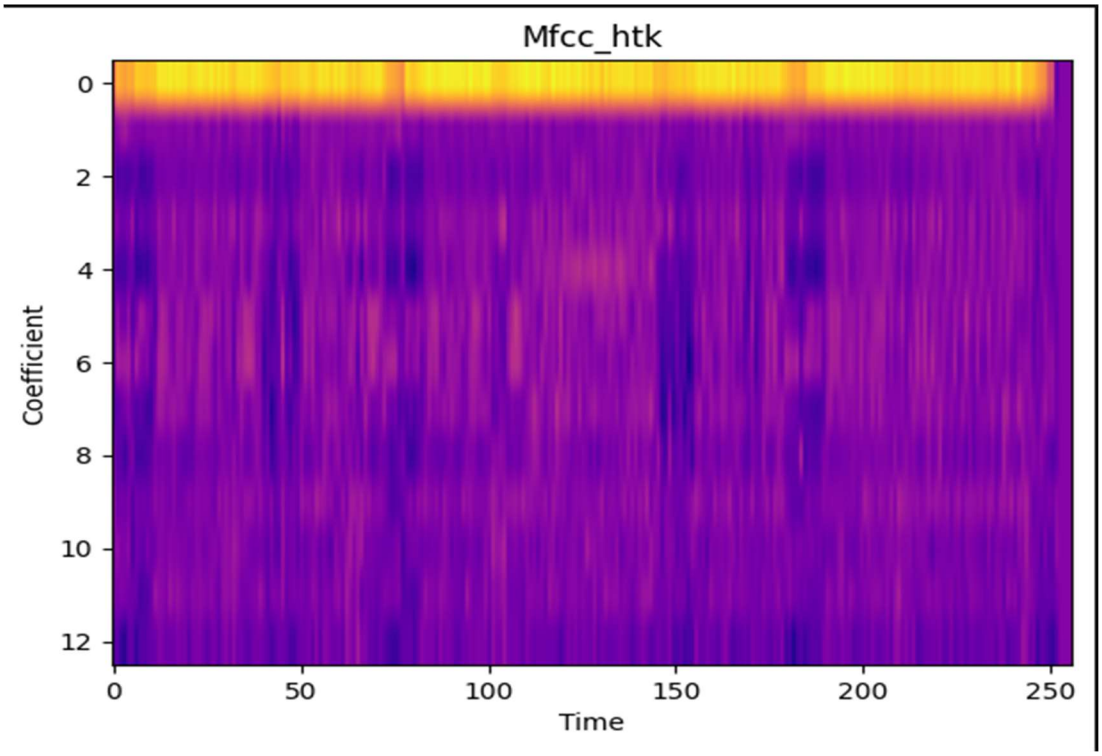


- Tempos: this a 2D array with two columns. The first column contains possible tempos and the second columns their probability (perhaps assigned to them by some algorithm). We calculated the weighted mean and consider this as the song's tempo for the rest of the analysis.

6. The next feature is a numpy array which can also be interpreted as an image, called mfcc_htk. This has a fixed number of 13 rows which however do not correspond to pitches (as in chroma_cens, crema and hpcp) rather than some coefficients. Columns are in this case as well timeframes.



We resize it to 13x256 shape.

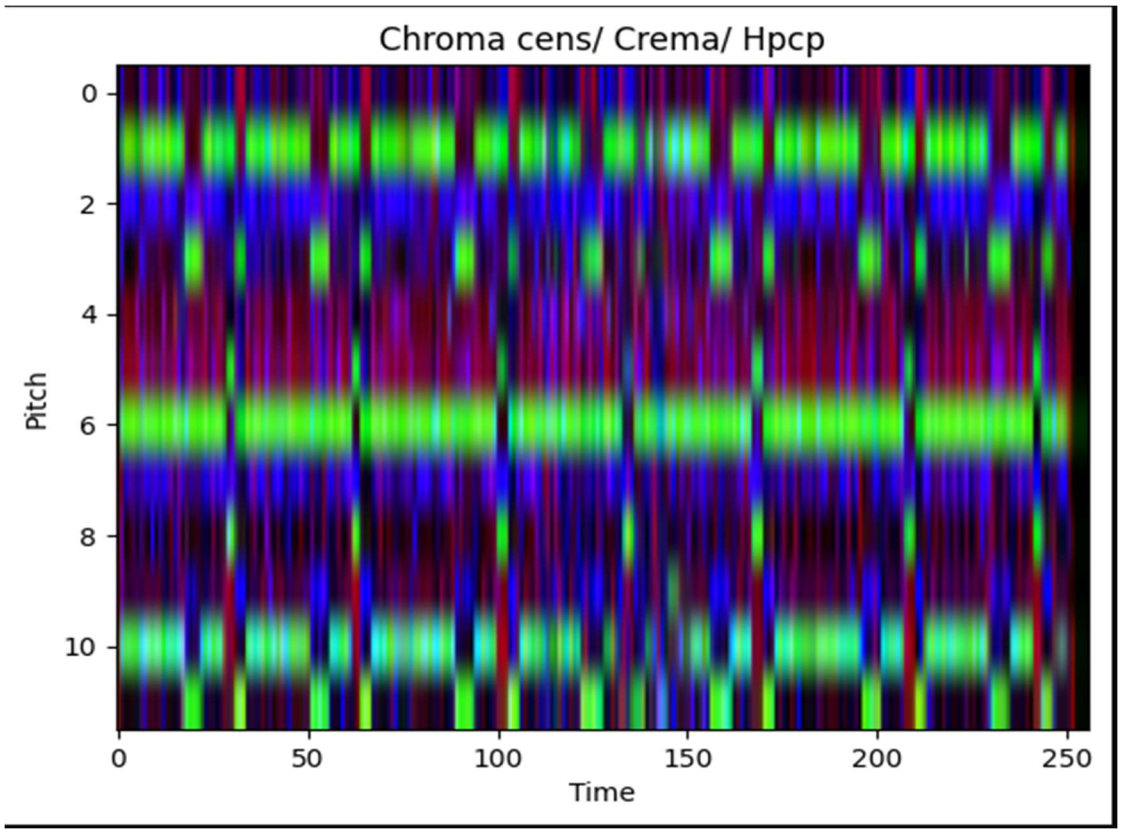


7. The seventh feature is a list of tags (e.g. rock, pop, instrumental etc) with their respsective degree of certainty (a float in range 0-1).
8. Label: covers of the same song as well as the original have same unique id (label)
9. Track_id: each performance has a unique id (we don't need that)

Methodology

We are going to implement a siamese network which is trained on pairs of samples (along with a label which indicates if the pair consists of similar or dissimilar songs) and learns to assign to each new pair of inputs a value that measures similarity. For this model we are going to use the following features as inputs:

1. As shown above the first three features (namely chroma_cens, crema and hpcp) represent the same thing in the two axis (pitches and timeframes). In order to make the model architecture more simple and to combine the information of the three features we stack them along the z-axis, creating a new feature with dimensions 12x256x3. An example of this image feature is shown below:

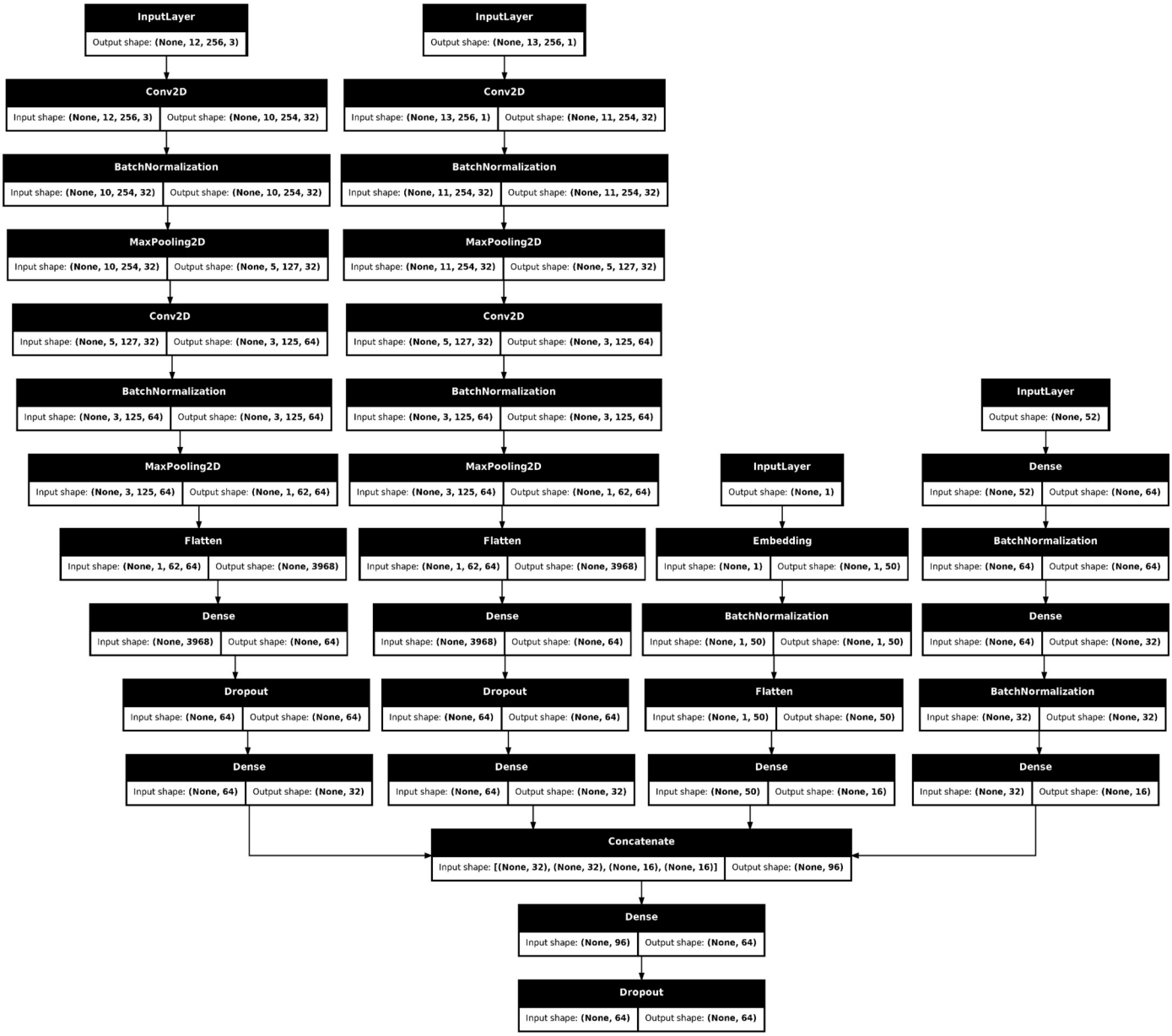


2. The mfc_htk image as explained, does not represent pitches so it was decided to be processed separately after being resized to 13x256.
3. The key/scale features are combined to one string (e.g Emajor) and are going be processed as text.
4. The last feature is an array containing numerical values:
    - From the madmom dictionary since the first three arrays contain similar information it was decided to select only one. At first the snovnf feature was selected but due to tits very high dimensionality (there were samples with up to 25000 values), finally the onsets array was used. In order to encode the most important information, since it is observed that the intervals between the onsets on each song are very similar (this is also confirmed by the peaks frequency in the novfn/snovfn plots) the mean value of the intervals was calculated and used as the first numerical feature (mean_onsets_interval).
    - As already mentioned, the weighted average of the candidate tempos is used as the second numerical feature.
    - The certainty degrees (probabilities) of the tags are used directly to capture more information on the songs.

    The numerical array contains 52 values (1 mean_onsets_interval, 1 tempo, 50 tags probabilities).

The sister networks of the siamese model process each feature of the sample separately and produce 4 embeddings (one for each feature) which are then concatenated to one total embedding for each sample. During training a pair consisting of two samples is fed to the model. The model produces one embedding for each sample and calculates the euclidean distance between them. This distance is the final output of the model. In order to train the model, a contrastive loss function is used which penalizes the model for assigning high scores to similar samples and for assigning low scores to dissimilar samples. For this reason the dataset is formatted such that the label for two covers of the same song (or for an original and its cover) is 0 and the label for two different songs (or covers of different songs) is 1.

The architecture of the sister networks is shown below.
For the two image features cnns are used followed by maxpooling and dense layers.
For the text key_scale feature embeddings are calculated at first followed by a dense layer and for the numerical features only dense layers are employed.
All the embeddings are concatenated and passed through a dense layer resulting to a final embedding of 64 values.



Conclusions/Improvements

Due to computational resources (lack of gpu and low ram) and time restrictions the model was trained on a small subset of 2000 samples. It did however start to learn to assign similarities correctly. Since we don't have labels regarding a similarity measure (we used 0 for similar and 1 for dissimilar) we check the mean similarity score for similar and dissimilar pairs to check the performance of our model. The mean score assigned to similar pairs was 0.405 while for dissimilar pairs it was 0.534. Obviously, it was expected to assign values closer to 0 to similar pairs and closer to 1 to dissimilar

ones. However considering the small amount of data and low hyperparameter tuning, it seems to be a quite promising approach and certainly the 0.13 difference indicates the network is producing lower scores for more similar songs.

This model could be improved by pretraining the sister network so it learns at first to extract more meaningful embeddings and then use it in the siamese model. This can be done by implementing an autoencoder with the encoder part identical to the sister network (shown above) and the decoder designed to reproduce the input. This way, the latent space representations would capture the most critical information of the input during the autoencoder training and then the similarity score of the siamese network could be more clear.

Considering training time, the main drawback of the current approach is the loading time and the size of the data, rather than the size of the model. In order to be able to train it in the full dataset there are two main improvements that could be done.

- Reduce even more the dimensionality of the images. At the moment 4 features are used per sample as described. Two of them are images and these are the ones of high dimensionality. They are resized to (12, 256, 3) and (13, 256, 1) respectively but still they require much lading time and memory. The reason was to maintain the important information but we could experiment with smaller shapes and check at which point the model starts to be actually affected in a negative way.
- Use a tf. Data generator to handle the dataset without loading all data at once. This way we can avoid memory issues when loading the data to memory for training the model.