A. AUDIO PREPROCESSING

In this paper, we used raw audio waveform data for training in both the pre-training and linear evaluation phases. The default audio sample rate for all experiments is 22 050 Hz, except for the sample rate experiment in Section C.2. The MagnaTagATune dataset contains monophonic 30-second audio fragments in MP3 format, sampled at 16 000 Hz. Some of the audio fragments originate from the same song. We reconstructed the original song by concatenating the fragments into a single file, to avoid occurances of fragments of the same song in the same batch of positive- and negative pairs, thereby ensuring i.i.d. data for training.

The audio files from the Million Song Dataset were obtained from the 7digital service, which provides stereo 30-second audio fragments in MP3 format sampled at 44 100 Hz.

All files were re-sampled to 22 000 Hz, 16 000 Hz and 8 000 Hz and decoded to the PCM format with `ffmpeg`, using the following command:

```
ffmpeg -i {input_file}.mp3 -ar {target_sample_rate} {output_file}.wav
```

This is the only preprocessing step that we performed before training.

B. DATA AUGMENTATION DETAILS

The default pre-training setting, which we also used for our best models, uses 8 audio data augmentations. Not all augmentations are necessarily applied to all inputs: each independent data augmentation is applied with a probability tuned during hyperparameter gridsearch. The most effective augmentations and their probabilities are presented in Section 4.3. The implementation details for each augmentation are provided below. We used the 'torchaudio-augmentations' Python package for all our experiments [45].

B.1 Random Crop

The audio is cropped with a number of samples $s \in \{20\,736, 43\,740, 59\,049\}$ for sample rates 8 000, 16 000 and 22 050 Hz respectively. To ensure that every sample in the batch is of the same size, the fragment's window we can crop from with original length S is adjusted to $S - s$.

B.2 Polarity inversion

The polarity of the audio signal is inverted by multiplying the amplitude of the signal by -1.

B.3 Additive White Gaussian Noise

White Gaussian noise is added to the complete signal with a signal-to-noise ratio (SNR) of 80 decibels.

B.4 Gain Reduction

The gain of the audio signal is reduced at random using a value drawn uniformly between -6 and 0 decibels. In our implementation, we use the `torchaudio.transforms.Vol` interface.

B.5 Frequency Filter

A frequency filter is applied to the signal using the `essentia` library [46]. We process the signal with either the `LowPass` or `HighPass` algorithm [47], which is determined by a coin flip.

For the low-pass filter, we draw the cut-off frequency from a uniform distributions between 2 200 and 4 000 Hz. All frequencies above the drawn cut-off frequency are filtered from the signal.

Similarly for the high-pass filter, we draw the cut-off frequency from a uniform distributions between 200 and 1200 Hz. All frequencies below the cut-off frequency are filtered from the signal.

B.6 Delay

The signal is delayed by a value chosen randomly between 200 and 500 milliseconds, in 50ms increments. Subsequently, the delayed signal is added to the original signal with a volume factor of 0.5, i.e., we multiply the signal's amplitude by 0.5. An example implementation of this digital signal processing effect is given below in Python using PyTorch:

```
import random
import torch
import numpy as np

ms = random.choice(
    np.arange(200, 500, 50)
)

offset = int(ms * (sample_rate / 1000))
beginning = torch.zeros(audio.shape[0], offset)
end = audio[:, :-offset]
delayed_signal = torch.cat((beginning, end), dim=1)
delayed_signal = delayed_signal * self.volume_factor
audio = (audio + delayed_signal) / 2
```

B.7 Pitch Shift

The pitch of the signal is shifted up or down, depending on the pitch interval that is drawn from a uniform distribution between -5 and 5 semitones, i.e., up to a perfect fourth higher or lower than the original signal. We assume 12-tone equal temperament tuning that divides a single octave in 12 semitones.

Pitch shifting is done using the `libsox` library, which is interfaced from the `wavaugment` Python library [34].

B.8 Reverb

To alter the original signal's acoustics, we apply a Schroeder reverberation effect [48]. This is again done using the `libsox` library that is interfaced from the `wavaugment` Python library [34].

C. ADDITIONAL EXPERIMENTAL RESULTS

C.1 Batch Size

The complexity of our contrastive learning approach increases with larger batch sizes, which may result in better representations. We pre-train from scratch until convergence with varying batch sizes and study its effect on the linear evaluation performance in Table C.1. While our smallest model already shows competitive performance compared to fully supervised models, the performance increased when using 96 examples per batch. Our largest model required more parameters to score consistently higher than our middle-sized model (≈25M parameters vs. 2.5M parameters). We hypothesise that the task of inferring the positive pair of 2.6 second long raw musical audio fragments, in a pool of 254 negative examples ($2 \times (128 - 1)$), may be simply too hard for a smaller encoder.

	Tag		Clip	
Batch Size	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
128	**89.7**	**37.0**	**94.0**	**70.0**
96	88.7	35.6	93.0	69.2
48	87.9	34.6	92.9	68.8

Table C.1: Effect of the batch size used during self-supervised training on the linear music classification performance.

C.2 Sample Rates

We show in Table C.2 that there is a marginal penalty to the final scores for the self-supervised models when re-sampling the audio to 8 000 Hz and 16 000 Hz respectively, which is in line with previous work [26]. Since re-sampling disturbs the frequency spectrum, we isolate its contribution by disregarding additional augmentations, i.e., only apply random cropping.

	Tag		Clip	
SR	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
8 000	84.8	29.8	90.6	62.9
16 000	85.5	30.4	91.0	64.1
22 050	85.8	30.5	91.3	64.8

Table C.2: Effect of varying the input audio's sample rate on the linear music classification performance.

C.3 Additional Hidden Layer and Training Duration

After pre-training with the self-supervised objective, we performed a linear evaluation to test the expressivity of the representations with a classifier of limited capacity. To further assess the representations' usability, we add a single hidden layer to our classifier and again measure the performance on the downstream task of music classification. The results of this experiment are shown in Table C.3 for linear evaluation (left of the forward slashes) as well as when a hidden layer is added (right of the slashes), for different pre-training durations measured in epochs.

Contrastive learning techniques also benefit from longer training compared to their supervised equivalent [17]. While larger batch sizes increase the pretext task complexity as shown in Appendix C.1, training longer increases the number of natural variations of the data, which is a diserable goal in representation learning [27], due to the random augmentation scheme. We pre-train from scratch until convergence and set the batch size to 96. Table C.3 also shows that increasing the self-supervised training duration improves downstream performance.

	Tag		Clip	
Epochs	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
10 000	88.7 / **89.3**	35.6 / **36.0**	93.2 / **93.5**	69.3 / **70.0**
3 000	88.5 / 88.9	35.1 / 35.5	93.0 / 93.3	69.2 / 69.7
1 000	88.3 / 88.6	34.4 / 34.9	92.3 / 93.1	68.6 / 69.2
300	87.1 / 87.4	32.7 / 32.5	92.0 / 92.0	66.6 / 66.7
100	86.4 / 86.6	30.9 / 31.3	91.3 / 91.3	64.1 / 64.6

Table C.3: Performance difference of a linear classifier and when a single hidden layer is added to the classifier on the downstream music classification performance, for different self-supervised pre-training durations.

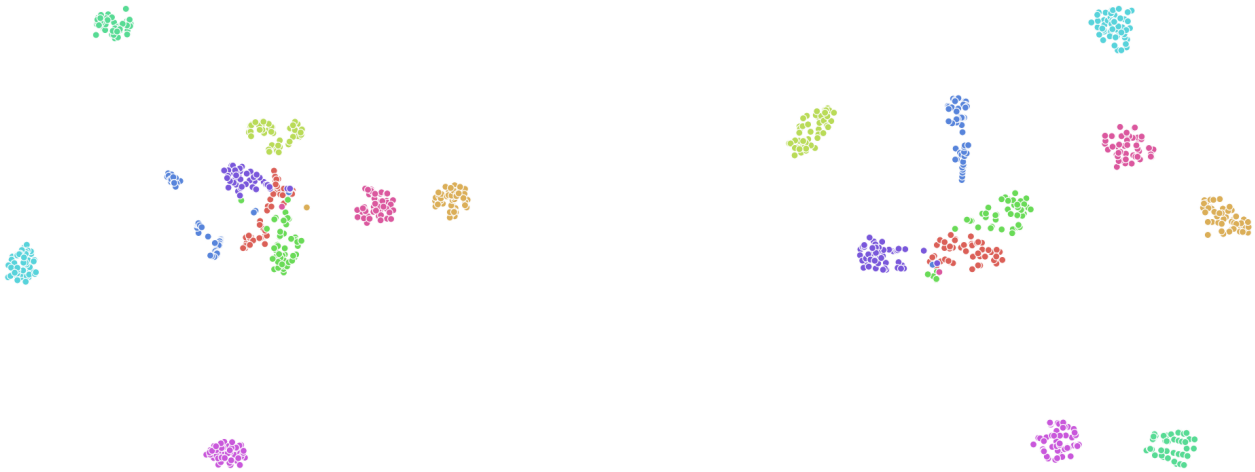
C.4 Qualitative Results

Figure C.1 shows t-SNE visualisations [49] of our best self-supervised models representations $\boldsymbol{h}_{\text{CLMR}}$ and $\boldsymbol{h}_{\text{CPC}}$, for a randomly set of music tracks from the validation set. We show that both self-supervised models can cleanly seperate the classes.

To get an understanding of what the self-supervised models capture from music, we show in Figure C.2 the magnitude spectrum of the learned filters of the sample-level convolutional layers (layers 1, 4 and 6) for CLMR and CPC, pre-trained on the MagnaTagATune dataset. We perform gradient ascent on a randomly initialised waveform of length 729, i.e., a value that is close to a typical frame size and also interacts conveniently with the convolutional structure of the encoder network, and subsequently calculate the magnitude spectrum. The x-axis plots the filter number, the y-axis the magnitude spectrum for a filter number. Lastly, we sort the plot by the frequency of the peak magnitude.

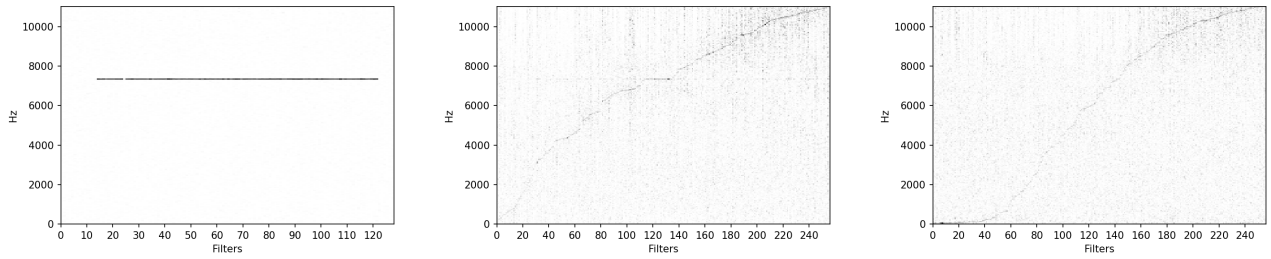
In CLMR, the first layer is sensitive to a single, very small band of frequencies around 7500 Hz, while in higher layers the filters spread themselves first linearly and then non-linearly across the full range. CPC shows a similar pattern in the higher layers, but shows a strong activation of two frequencies that span an octave in the first layer. Conversely, the filters of the supervised-trained encoder have a non-linearity that is found in frame-level end-to-end learning [36], as well as in perceptual pitch scales such as mel or Bark scales [50, 51].

Figure C.3 shows the sorted tag-wise ROC-AUC scores for the top 50 tags in the MagnaTagATune dataset, reported for linear evaluation of the trained self-supervised CLMR and CPC models, and the fully end-to-end-trained supervised model. We show that no single tag loses more than 4% ROC-AUC when trained using self-supervised pre-training and fine-tuning is only performed with a linear classifier, as compared to the supervised benchmark.

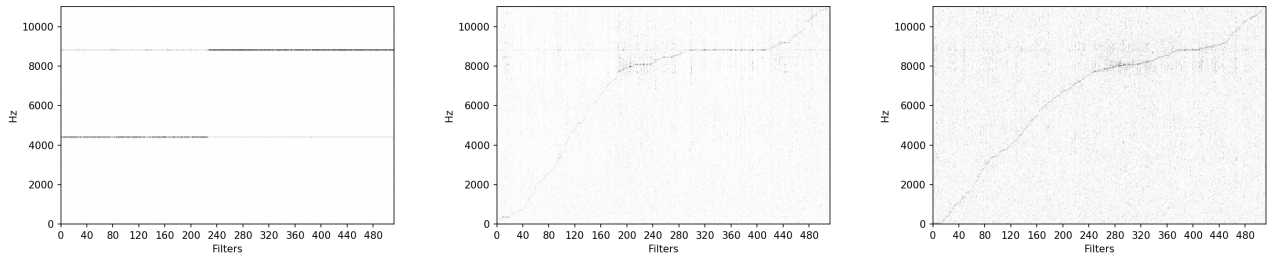


(a) $\boldsymbol{h}_{\text{CLMR}}$ (b) $\boldsymbol{h}_{\text{CPC}}$

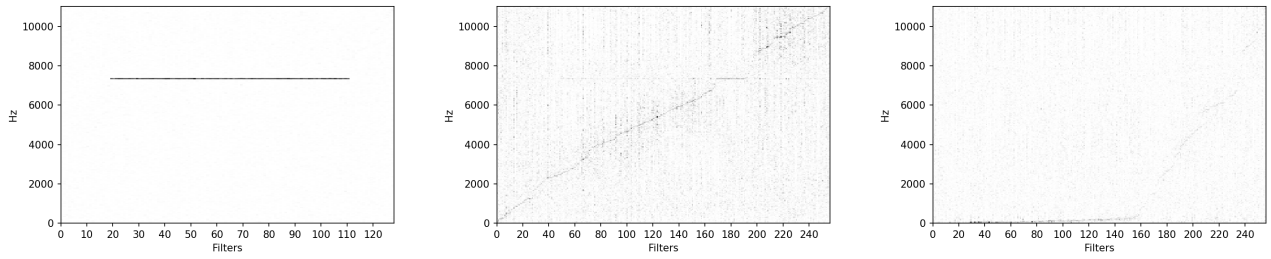
Figure C.1: t-SNE manifolds of the hidden vectors of music audio from a subset of 10 music tracks, i.e., in this case classes, from the validation set. Each point represents a 2.67 second long music fragment belonging to a music track.



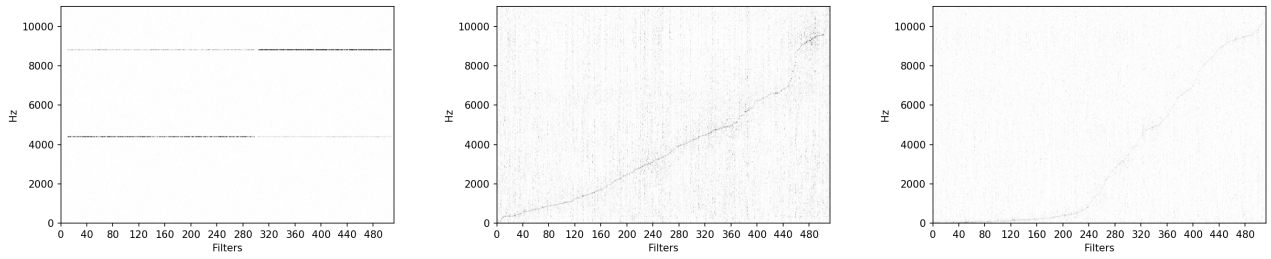
(a) $\text{CLMR}^{(1)}_{\text{MTAT}}$ (b) $\text{CLMR}^{(4)}_{\text{MTAT}}$ (c) $\text{CLMR}^{(6)}_{\text{MTAT}}$



(d) $\text{CPC}^{(1)}_{\text{MTAT}}$ (e) $\text{CPC}^{(4)}_{\text{MTAT}}$ (f) $\text{CPC}^{(6)}_{\text{MTAT}}$



(g) $\text{CLMR}^{(1)}_{\text{Billboard}}$ (h) $\text{CLMR}^{(4)}_{\text{Billboard}}$ (i) $\text{CLMR}^{(6)}_{\text{Billboard}}$



(j) $\text{CPC}^{(1)}_{\text{Billboard}}$ (k) $\text{CPC}^{(4)}_{\text{Billboard}}$ (l) $\text{CPC}^{(6)}_{\text{Billboard}}$

Figure C.2: Normalised magnitude spectrum of the filters of the self-supervised models in the sample-level convolution layers, sorted by the frequency of the peak magnitude. Gradient ascent is performed on a randomly initialised waveform of 729 samples (close to typical frame size) and its magnitude spectrum is calculated subsequently. Each vertical line in the graph represents the frequency spectrum of a different filter. The first three images are taken from a pre-trained, converged CLMR model, the last three from a CPC model, on the MagnaTagATune or Billboard datasets

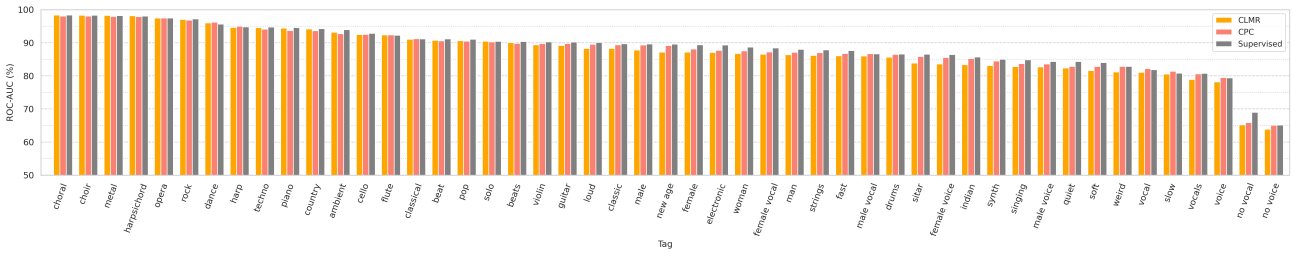


Figure C.3: Tag-wise ROC-AUC scores for the top-50 tags in the MagnaTagATune dataset, reported for linear, logistic regression classifiers trained on representations of self-supervised models CLMR and CPC, and compared to a fully supervised, end-to-end SampleCNN model.