Part I.

State of the art of research in audio music tagging

A word on the music tagging task and learned signal processing

This section will be focused on exploring Audio preprocessing techniques and machine learning models towards music tagging. This preface does not delve into detail into the specifics of the models explored later on but rather proposes a novel view of machine learning applied to audio as a learnable framework for more black-box audio processing blocks and serves as context with regards to the use of these music tagging models. Though it might be intuitive to consider these as two disctinct tasks of the same pipeline, we propose a different outlook : rather than exploiting learned representations of audio such as spectrograms through a learned model that only serves the purpose of "analyzing" these coefficients in a learned way, we propose that these machine learning models can be considered in and of themselves a learned representation extractor with regards to audio signals.

For instance, consider the canonical music tagging pipeline, which we can summarize here. usually, audio signals of fixed length are used to generate mel-spectrograms in a first preprocessing step. These melgrams are passed through a convolutional neural network which learns the tagging coefficients directly (*nota : As we will see later on, some audio models directly use raw audio to learn tagging coefficients*). A traditional Music tagging task would look like this (Figure 8.2)

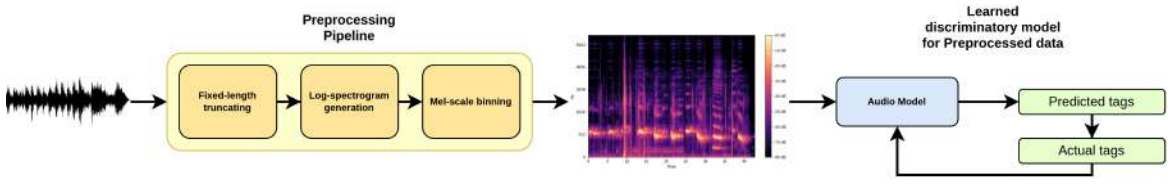


Figure 2.18.: traditional audio tagging task pipeline

In a traditional view, the preprocessing pipeline and the inference and learning pipeline are mostly disconnected. We propose a different view, that Audio models are learned latent representations of audio data based on the extracted data. Only the last few layers of the concerned deep learning models can be considered as classifiers used to discriminate from a learned representation of the audio which is computed in a full pipeline as opposed to before, two different pipelines (figure).

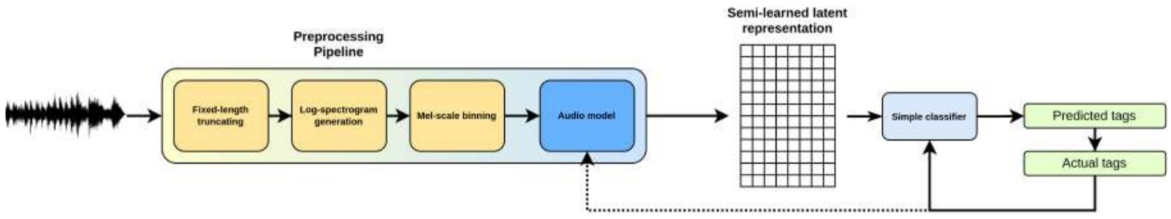


Figure 2.19.: new audio tagging task pipeline

Essentially, we intuite that past a certain point in training, contribution to the higher-level layers of the audio models are minimal compared to shifts in the weights of the simple classifiers which serve as latent representation discriminators based on the data extracted from the spectrograms (or other representations). This is corroborated by the value of pre-training (exhibited for instance in [4], [36]) in learning useful music representations which can be adapted for any downstrem task. By training directly on a downstream tagging-task we learn to generate a useful representation for the task of music tagging, which can eventually be generalized to other audio-oriented tasks.

An audio model appended after a traditional preprocessing pipeline can thus serve as what we now call a **latent representation extractor**, and generate what we now call **learned latent representations**. Much as spectrograms are frozen and deterministic representations of an audio signal relating to spectra and temporal evolution, Frozen audio models without their classifier heads are determinstic representations of an audio signal. We argue that the value of these learned representations can be taken out of the Music Information Retrieval domain to other audio domains, thus showing their use in other signal processing tasks (Much like machine learning has helped determinstic methods in fluid mechanics and Computational Fluid Dynamics for instance [42])

3. Audio preprocessing techniques

The basic raw audio data is in waveform format, or an array of floating samples with a given sampling rate (notated hereforth *sr*.) Preprocessing as a whole for audio includes various techniques which are applied to this raw audio before being fed through an inference model. This can include **segmenting, applying a transformation so the audio is in a new representation space, log-scaling, etc**

We do not consider data augmentation in the first part of this section as this is more relevant to model training. However, audio augmentation is an ensemble of techniques (reverb, delay, saturation, noise...) used to shift the input audio in the representation space to prevent overfitting in the mode during training. Preprocessing in and of itself is the pipeline applied before training which aids in giving the model an appropriate representation of the data.

3.1. Signal-based techniques

Signal-based techniques are defined as techniques applied directly to the raw audio signal (array of 1D floating point values) with the goal of making them more viable for machine learning tasks.

3.1.1. Filtering and filter banks

Filtering a signal a signal applies a convolution operation the weights of which reduce or augment the spectral composition of a signal at various frequencies [11], [7]. Common techniques for music classification tasks are **20Hz-20kHz band pass filters** [1] applied on all audio waveforms to conform to the human hearing range. For instance, when using data labels produced by humans, considering spectra components outside of the human hearing range does not make sense as they were not taken into account when producing labels.

Other, more case-specific techniques might be of use when considering specific applications . A few examples:

- Filtering for a given band of frequencies for instrument classification (for instance, filtering out lows for string instrument classification)
- Filtering out rumble frequencies (20-50Hz) and shimmer frequencies (19kHz - 20kHz) for percussive instrument classification
- Sound event classification (example Urban8K dataset [34] audio event classification task) where interesting frequencies and/or event spectral fingerprints might not lie within the human hearing range.

Librosa provides a bank of available filters at the documentation for librosa. Kapre also provides a filter bank layer for easier application during training and inference, as well as GPU-processed filtering.

3.1.2. Resampling

Sample rate represents the rate at which information is readily available within an audio signal. **The industry standard for sample rate is 44.1kHz**, which allows for a Nyquist frequency slightly above the human hearing threshold. However, it is possible to resample a signal to any given frequency.

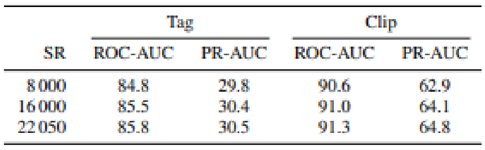
While this reduces space constraints and computation times for model inference (by effectively reducing the size of the input vector), it also is congruent to loss of information. For instance, resampling a 44.1kHz audio clip to 16kHz would lead to loss of more than half the data points.

In practice, the need to resample might arise when dealing with noisy real-world datasets. For instance, in **Rethinking CNN Models for Audio Classification** [30] The various datasets used to benchmark the model

[1] A note on musical recordings : Usually, mastering engineers already apply band-pass filtering to their master track: removing shimmer and rumble frees up space for other frequencies in a song which are more audible and agreeable to the human ear. However, this is not necessarily an industry standard, just good practice. So, while applying a band-pass might be redundant in some musical application cases, there usually is not 100% certainty that it will be.

on any given task have varying sample rates between themselves and between clips of the same dataset. The Urban8K dataset has sample rates varying between 16kHz and 44.1 kHz, and is uniformized to 22.5kHz. [2]

In terms of influence of sample rate on model performance, it is often minimal, and evolves as expected : a lower sample rate leads to lower performance due to the loss of information, be it on raw audio or transformed spectrograms with lower sample rates. (See table below taken from **Contrastive Learning of Musical Representations** [37]



	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

Figure 3.1.: Performance evolution with sampling rate for contrastive learning of musical representations [37]

The performance hit is marginal, yet significant for industrial applications. So it is important to bear in mind sample rate and not dismiss it right away without at least checking for adjustment necessity.

Conceptual approaches to sample-rate conversion include: converting to an analog continuous signal, then re-sampling at the new rate, or calculating the values of the new samples directly from the old samples. The latter approach is more satisfactory, since it introduces less noise and distortion.[3] Two possible implementation methods are as follows:

If the ratio of the two sample rates is (or can be approximated by) a fixed rational number L/M: generate an intermediate signal by inserting L 1 zeros between each of the original samples. Low-pass filter this signal at half of the lower of the two rates. Select every M-th sample from the filtered output, to obtain the result. [29] Treat the samples as geometric points and create any needed new points by interpolation. Choosing an interpolation method is a trade-off between implementation complexity and conversion quality. Commonly used is the windowed sinc function for audio. The two methods are mathematically identical: picking an interpolation function in the second scheme is equivalent to picking the impulse response of the filter in the first scheme. Linear interpolation is equivalent to a triangular impulse response; windowed sinc approximates a brick-wall filter (it approaches the desirable "brick wall" filter as the number of points increase). The length of the impulse response of the filter in method 1 corresponds to the number of points used in interpolation in method 2.

Method 2 will work in more general cases, e.g. where the ratio of sample rates is not rational, or two real-time streams must be accommodated, or the sample rates are time-varying.

Both kapre [9] and librosa [27] provide resampling functionality for audio signals. The default filtering window used for the resampling interpolation algos used by both kapre and librosa is a kaiser window, which can be defined as :

$$w_0(x) = \begin{cases} \frac{1}{L}\frac{I_0 \pi\alpha\sqrt{1-(2x/L)^2}}{I_0 \pi\alpha}, |x| < L/2 \\ 0, |x| > L/2 \end{cases} \tag{3.1}$$

where:

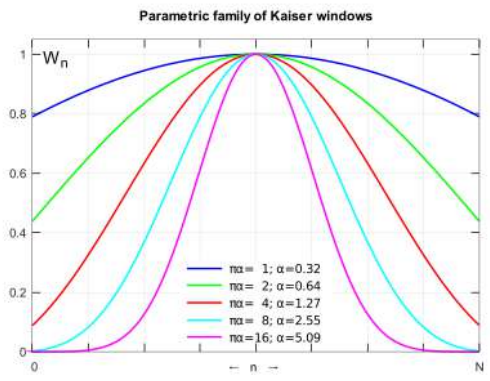
- I_0 is the zeroth-order modified Bessel function of the first kind,
- L is the window duration, and
- α is a non-negative real number that determines the shape of the window. In the frequency domain, it determines the trade-off between main-lobe width and side lobe level, which is a central decision in window design. Sometimes the Kaiser window is parametrized by β, where $\beta = \pi\alpha$.

The shape of which can be seen figure 3.2a. The fourier transform of this window is given by:

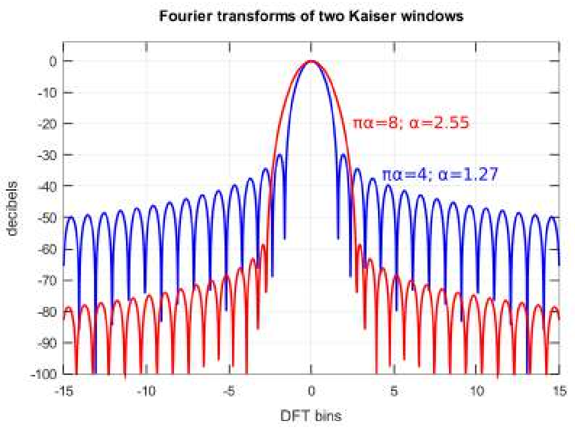
[2]Note on musical applications: As there is an industry standard in place on streaming platforms, it is safe to say that most clips will have at least a sample rate of 44.1. However, high quality audio (96kHz is possible, as well as 122kHz) also exists and so a check is necessary to ensure fixed-length input to the model even if no space-saving related resampling is considered.

$$W_0(f) = \frac{\sin(\sqrt{(\pi L f)^2 - (\pi\alpha)^2})}{I_0 \pi\alpha \sqrt{(\pi L f)^2 - (\pi alpha)^2}} \tag{3.2}$$

The shape of which is shown figure 3.2b with various parameters:



(a) Kaiser windows for various parameters



(b) Kaiser window spectra for various parameters

And the equivalent window function for a discrete signal (digital audio signal processing) is given by :

$$w(n) = L w_0(L/N(n - N/2)) \tag{3.3}$$

having $N+1$ the length of the window. both kapre, librosa, and torchaudio [55] use fast and best variations of the kaiser window to conduct resamplig. Interpolation and decimation to obtain the target sample rate are both conducted using the discrete kaiser window and since the default resampling algorithm is used in the entirety of state of the art, we assume that use of the default kaiser window in this work is appropriate and does not require further investigation.

3.1.3. Mono

Mono is the operation of time-averaging the signals of the two channels of stereo signals (L-R). This reduces the input dimensionality of the signal, at the cost of music-relevant information. Indeed, stereo spacing is often an artistic choice and is relevant when humans are listening to music. So, it follows that reducing input to mono distances the model from the artistic vision of the mixing engineer, which is integral to track characteristics.

Canonical music classification datasets adopt various approaches to this issue, as for instance the **GTZAN [41] and UrbanSound8K [34]** datasets provide mono samples. The FMA dataset [10] provides Stereo clips. Both for loading and other operations, Librosa and Kapre provide stereo support. [52] does not provide emphasis on the use of either mono or stereo.

Per the difficulty of finding any good practice recommendation of mono vs stereo input clips in the current literature, it can be said that the choice is rather empirical, or even model-based. If the model used for the classification task does not support multichannel input, then mono is adopted. However, intuition dictates that when available, stereo should be used, as perhaps spatial features can be learned by the model. At any rate, librosa proposes a handy mono function should the need arise to make a track mono for analysis, though kapre does not:

```
import librosa
y = librosa.to_mono(y)
```

3.1.4. Smoothing / Time averaging

Smoothing, or time-averaging a signal, is a preprocessing technique used to smooth out the envelope of an audio signal, and thus generate a new vector with the average envelope of the signal over the given time frame. This is mostly useful for onset detection though, and has little application in tagging tasks, bar its implication in tempo and other rythmic features detection (these modules are built into librosa as will be seen later on).

Obviously, high-frequency details are lost when smoothing out the signal.

3.2. Spectral techniques

The general purpose of spectrograms is described in Section 3.2.1. They are 2D representations of an audio signal which show the energy distribution of the signal along both the time and frequency axis, making them biologically relevant feature maps of the signal.

Their setup can vary, and modern classification tasks rarely use pure STFTs as input. More performant models have been elaborated by using log-frequency spectrograms, log-magnitude spectrograms, MFCCs and Mel-spectrograms.

3.2.1. Discrete Fourier Transform & Short Time Fourier Transform

Discrete Fourier transform and short time Fourier transform are the respective time-unaware and time-aware analogs of the Fourier transform. this transformation allows for computation of the power distribution of the given sample over the sound frequency spectrum.

Discrete Fourier Transform (DFT) DFT is used to extract spectral representations from a discrete digital signal. recall the expression of the discrete fourier transform of a signal y into its spectrum Y:

$$Y_k = \sum_{n=0}^{N} y_n e^{\frac{-2i\pi nk}{N}} \tag{3.4}$$

based on state of the art, the DTFT is very rarely used in music classification tasks (no papers mention it). This makes sense intuitively as DFT does not represent the evolution of the spectral components of the signal over time, but only over the globaility of the signal, and il also non-discriminant towards unsignificative values. It is relatively high-dimensional data for not much information. The following figure shows the DFT for one of the tracks in the FMA dataset [10] (Figure 3.3) :

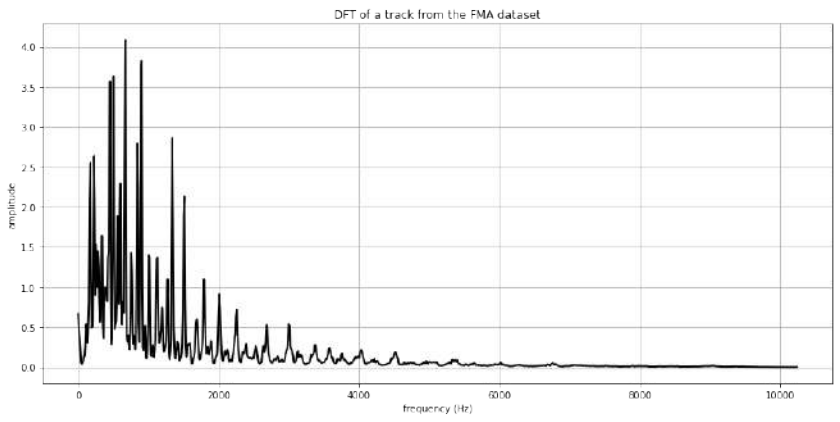


Figure 3.3.: DFT of track from the FMA dataset

Short time Fourier transform (STFT) The rawest kind of Spectrogram, with a linear frequency scale and linear amplitude scale. it's a 2D representation of the signal which is used as input for convolutional neural networks towards music classification tasks. However, it has since been dethroned by the Mel-spectrograms due to the more biologically-plausible representation it provides.

Its strengths stem from the fact that it is a representation of the evolution of the spectral components of the signal over time. Looking at the expression for DFT, the spectral components for the signal are computed over the whole signal. To paliate this, Gabor ([12]) proposes computing the DFT of the whole audio over chunks of the signal to get timewise evolution of the spectral components : This can be achieved through multiplying the signal by a window function centered around various points of the signal. To mitigate artifacts created by analyzing directly-adjacent chunks of the signal, the chunks of the signal are taken as overlapping. The following is the expression of the STFT for a discrete signal with a window function w

$$Y(k,m) = \sum_{n=0}^{N-1} w(n) y(n+mH) e^{\frac{-2i\pi kn}{N}} \tag{3.5}$$

Where $y(m+H : N-1+m+H)$ is a segment of y, H is the hop length, in samples (by default in python audio libraries, half of the $N_f ft$), w is the window function. m is the frame index, k is the frequency bin index.

STFTs are often computationally expensive as they provide high dimension 2D representations (For instance, the following STFT spectrogram computed for the same track as figure 3.3 (figure 3.4) with a $n_f ft$ of 2048, a H of 1024 and a Window length of 1024 samples, with a $16kHz$ sample rate, is og shape (1025, 9258), which yields 9.5M samples for 3 minutes of audio.

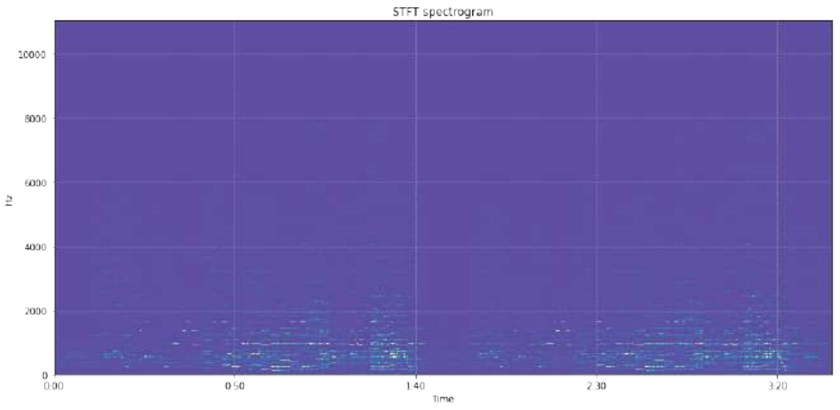


Figure 3.4.: STFT of track from the FMA dataset

The number of FFT bins dictates the spectral resolution, at the expense of temporal resolution. To alleviate this trade-off and prevent artifacts in the fft generation, the use of windows is necessary: A window is a function which multiplies the convolution boundary to smooth out the obtained spectra. Some options for window functions are the following:

- Hanning:

$$w_0(x) = \begin{cases} 1/L\cos^2(\frac{\pi x}{L}), |x| < L/2 \\ 0, |x| > L/2 \end{cases} \tag{3.6}$$

- Hamming
- Black

The default used by many studies and notably the **Librosa** module is the hanning window, which is sufficient for most audio processing applications. The stft function is readily available in librosa and kapre. [3]

[52],[7], [8] [6], [30] All mention that MFCCs outperform STFTs and melgrams outperform MFCCs by a respectable margin. STFTs are seldom used in recent studies per their large size and usually lesser performance.

[3]Note : the STFT is a dual representation : magnitude and phase. Often in classification tasks the phase is neglected (especially when dealing with convolutional neural networks). However, in tasks relative to speech and music generation, phase is a non-negligible component of the signal (eg : in speech generation tasks where phase is neglected, the generated vocals come out flat and robotic).

3.2.2. Mel Frequency Cepstrum Coefficients (MFCC)

Mel Frequency cepstrum coefficients (MFCCs) are a set of features which can be used to describe the overall spectral envelope of a signal. They are interesting in terms of audio classification tasks as they provide a low-dimension representation of the signal. They are based on the mel frequency scale which mimics human perception of frequencies: the following figure (Figure 3.5) shows the mapping function from Hz to mels [2]:

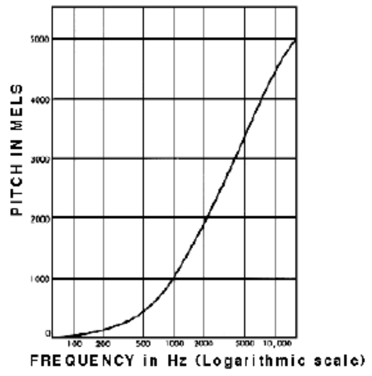


Figure 3.5.: Mel Scale for Audio Frequencies

The calculation of the pitch in mels based on the frequency f is defined as:

$$f_{mel} = 1000\frac{\log_{10}(1 + f/1000)}{\log_{10}(2)} \tag{3.7}$$

The coefficients are obtained by taking the outputs of N_f log-scale triangular filers of a signal into Mel-frequencies and applying a discrete cosine transform. The formula for the cepstral coefficients is given below:

$$c_i = \sum_{n=1}^{N_f} S_n \cos\left(i(n-0.5)\frac{\pi}{N_f}\right), i \in [0...L] \tag{3.8}$$

Where L cepstral coefficients are wanted, and S_n is the log-energy output of the n^{th} triangular filter. The MFCC calculation using librosa [27] for the track in figures 3.4 and 3.3 is given in the following figure (3.6)

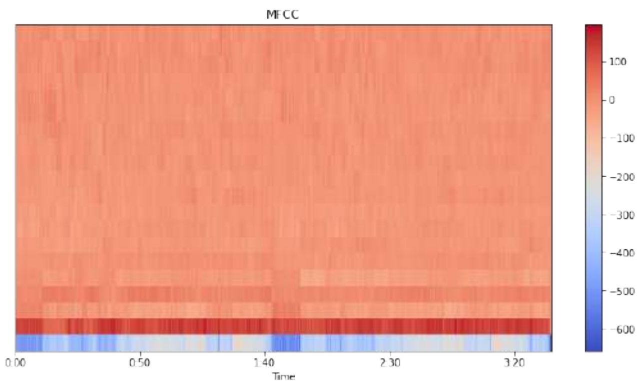


Figure 3.6.: MFCCs for the track from the FMA dataset

in [6], auto music tagging is conducted using among others MFCCs as features. MFCCs have since been outperformed by Mel spectrograms in all music classification tasks, as we will see in the next section.

3.2.3. Mel-spectrograms

Mel-spectrograms are a combination of STFT and MFCC, in that they are obtained by averaging the STFT values over mel-bins, which yields a lower-dimension representation which is more adapted to human perception of sound. Melgrams are the default feature input for convolutional classification models, especially on tagging. [7] [8] [6] [30] All use melgrams as their input feature, and more specifically [6] [8] [7] [48] use them as part of a tagging task, which is relevant to our use case. The following figure (Figure 3.7 shows the mel-spectrogram for the track used in all other figures:

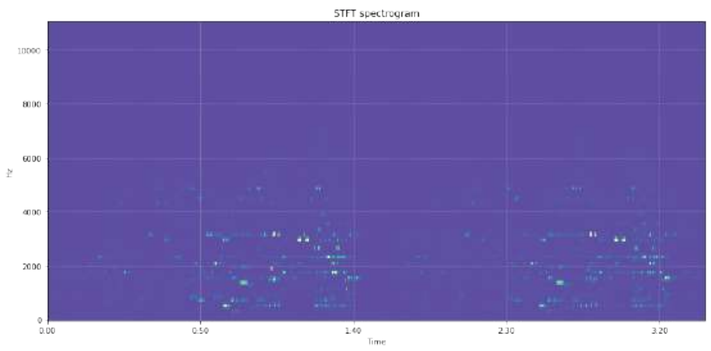


Figure 3.7.: Melgram of track from the FMA dataset

For instance, this melgram of the same track, computed using the same FFT hyperparameters as in 3.4 yields a feature map of shape (128,9285), which represents a 90% reduction in input size dimension (much better for storage, inference time). Furthermore, Melgrams have been shown to outperform or reach the same performance as STFTs in [7] (see figure 3.8):

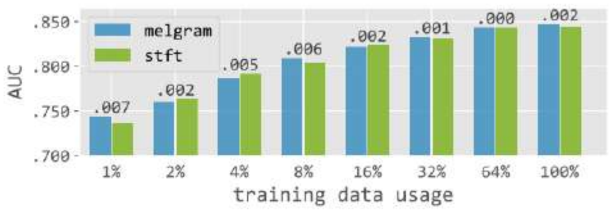


Figure 3.8.: Performance of mel spectrograms versus STFT on tagging tasks versus training data

And in [6] as seen in the following table:

Method	Input	ROC-AUC
FCN-3,	mel-spectrogram	.852
FCN-4,	mel-spectrogram	.894
FCN-5,	mel-spectrogram	.890
FCN-4,	STFT	.846
FCN-4,	MFCC	.862

Table 3.1.: Input performance on music tagging as seen in [6]

Implementations of melgrams are readily available in both Librosa and Kapre. Per their ease of use, low dimensionality and high performance, mel-spectrograms are often considered to be the benchmark for industry applications in most music classification tasks and specifically music tagging tasks [4]. However, we scrutinize some

[4]Furthermore, it is possible to generate stacks of spectrograms and melgrams as inputs by producing various spectrograms with varying hop widths/window sizes. This allows to capture features at various time and frequency scales, which can be beneficial to the model learning them. However, no studies compare the results obtained for such variations.

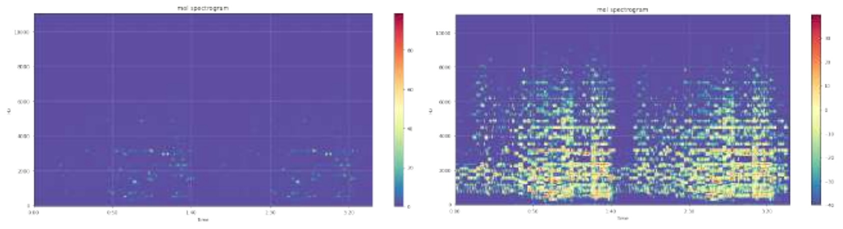
additional prepcrocessing techniques in Annex A.2 (**Constant Q melgrams, Chroma Variants, Spectral Envelopes, Tempograms**).

3.3. Scaling techniques

It is well known that data scaling is essential for good performance of most machine learning models. This prevents gradient explosion for skewed data distributions. In simple classification systems, all features are independently scaled to reach unit normal distribution. In computer vision, image values are scaled for R,G,B channels. For audio preprocessing, the data for STFTs can be scaled using log-magnitude scaling:

3.3.1. log-melgrams

Due to our non-linear perception of Loudness [43], it makes more sense from a biological standpoint to represent loudness in 2D time-frequency inputs as decibel values, i.e taking the log of the modulus of the STFT transform. Here are two Spectrograms representing the same signal, one of them being log-scaled and the other one not (figure 3.9):



(a) Raw mel spectrogram (b) Log-scaled mel spectrogram

Figure 3.9.: raw and log-scaled melspectrogram

As can be seen, the magnitude scale is highly compressed, which usually represents a significant gain in performance for a machine learning model, especially regarding image data. The reduced scale prevents gradient explosion. The following shows the distribution of the binned magnitudes for the non-log scaled stft and for a log-scaled stft (Figure 3.10) taken from [7]. These histograms are averaged over 100 songs from the million song dataset.

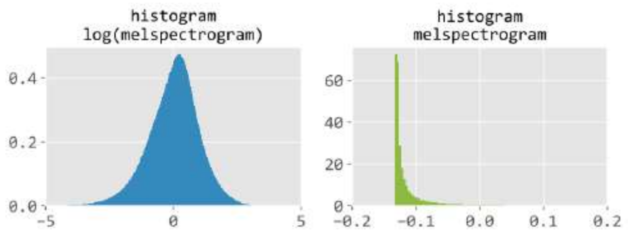


Figure 3.10.: Log-scaled and non-log scaled bins for the melgrams in [7]

In terms of performance on audio tagging tasks, [7] shows that log-scaled melgrams always outperform regular melgrams, the same goes for log-scaled STFTs versus standard STFTs (Figure 3.11).

All studies mentioning melgrams use log-scaled melgrams and some even explicitly state that log-melgrams are the norm. There is no upside to using linear scaled melgrams at all regarding audio classification. Furthermore, it is also possible to apply α-log scaling to the magnitude of the melgrams, which is obtained by applying the following transformation :

$$X \to \log(X + \alpha) \tag{3.9}$$

the exploration of the epsilon parameter however, is non-existent. Studies arbitrarily apply it as a large or small number, or not at all. We decide to keep it as 0. Librosa and Kapre both implement handy default

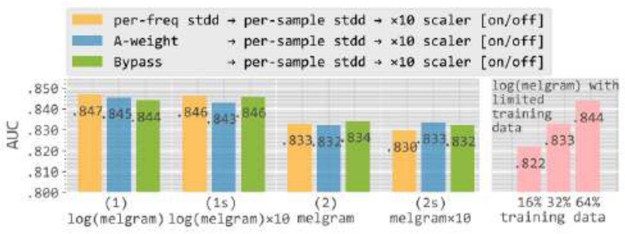


Figure 3.11.: performance of non-scaled STFTs and melgrams vs scaled STFTs and melgrams on music tagging tasks [7]

behaviours regarding this log-scaling, which is set to true by default. Furthermore, it is possible to apply 0-centering through the appropriate function parameters.

3.3.2. Frequency scaling

Per human's non-linear perception of frequencies (we naturally perceive higher frequencies as louder, frequency weighting can provide marginal performance gains when looking at classification models. There are two main techniques [7]

- **Scaling per frequency band** : Also known as spectral whitening. Each frequency band is scaled using mean and standard deviation in the frequency band. This yields a flat spectrum which resembles that of white noise (hence the name).
- **A-weighting** International systems exist to weigh spectra magnitude values as a function of the frequency for various applications. : A-weighting attempts to represent human perception of sound (higher frequencies are felt as louder) C-weighting is for industrial applications and noise regulation laws, and dips off in the very low end (rumble) and very high end (shimmer) Z-weighting is a flat filter (See figure 3.12) [47].

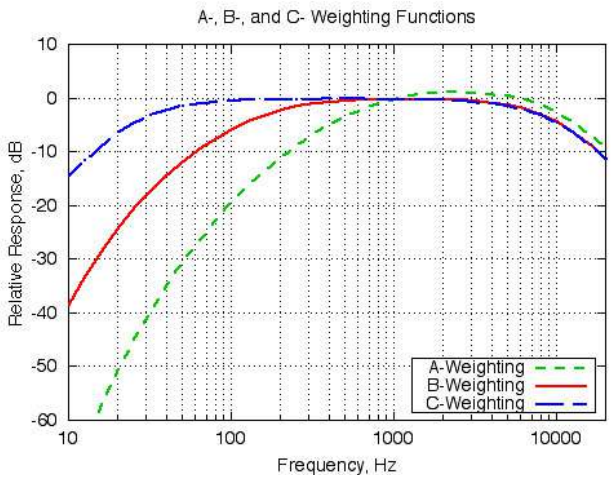


Figure 3.12.: International standards for A,C,Z weighting [47]

Such filters can be applied to the signal as part of preprocessing. However, the performance gain depends on the context and is often marginal. Figure 3.11 shows ROC-AUC performance in [7] for the two described techniques as well as bypassing frequency weighting. There is no true advantage in the case of music tagging to applying frequency weighting to the signals.

3.4. Splitting

3.4.1. Fixed length input splitting

Often, the input to audio classification models is in the range of the 5-10 seconds [7], [52] length (which still represents around 200000 samples). Some musical datasets have gone up to 30s ([41]), but few feature whole length songs.

It can be difficult and not necessarily task-relevant for models to capture relevant features at whole song-levels. This is why a common preprocessing technique is to split the audio clip into fixed input segments which are then voted upon or average (in what is called multi instance learning) to determine the class of the whole sample or can be stacked to create a multi-dimensional vector (think RGB channels for computer vision).

Additionally, splitting the input into different sample sizes (2s, 10s and 30s for instance) can be beneficial towards capturing audio features at different time scales. This is effectively increasing the receptive field of the network.

It is shown in [52] that the maximum length for inputs in current music classification models is 30s, with many being around 3-5s [6], [8], [32], [22], [49]. Splitting adds another advantage, which is that of data augmentation : one 10-second long song yields 5*44100 training samples, which makes for huge training potential with minimal labeling requirements (it does require high storage space though).[5][6]

3.4.2. Padding

Obviously, all audio clips are not of perfect multiple length regarding the desired input length. Zero-padding to attain fixed input length is a common technique in other domains as well (sequence prediction, computer vision, NLP...) and so we do not elaborate on it here, as its effects are similar to those in those other domains and are presumed to be inoffensive.

3.5. Audio augmentation techniques

As mentioned previously, audio augmentation techniques are a set of techniques used to shift the audio sample in the representation space : Audio augmentation, a subset of techniques of general data augmentation, changes the input slightly when training the model to prevent the model learning on the exact distribution of the training set - acting as a regularization technique. Here we summarily discuss audio augmentation techniques and their use in state of the art.

Random cropping and filtering have been discussed previously in the context of pure audio processing. They can also, however, be used as audio augmentation techniques [52]. By random cropping audio inputs, the representation of a given track changes every time it is fed to the model. Similarly, using high-pass and lowpass filters on a track with varying probabilities acts as a disruptor of the original signal (different features are seen. This is used in [36] and [48] using randomized filter cutoffs.

Delay and reverb Delay and reverb are techniques used to modify the acoustic space of the signal. Delay is applied my delaying the signal by a randomly generated delay time (in ms) and adding the delayed signal multiplied by a reductive factor to the original signal. Reverb is generated using impulse response's room size, reverbation and damping factor, drawn from a uniform distribution. Again, these are used in [36] [24] [48]

Polarity inversion and pitch shifting relate to the frequency characteristics of the waveform. Polarity inversion of the waveform is obtained by multiplying it by -1, and pitch shifting is applied by common algorithms provided by librosa, torch augmentations, audiomentations, etc. Usually, pitch shifting occurs by increments of semitones in the 12-tone equal temperament range within 1 octave of the original.

[5] An extension of this is to split the song into relevant segments (for instance, verse chorus and bridge) as from an artistic standpoint it makes sense to distinguish them, so to does it from a tagging standpoint.

[6] Removing the beginning of songs can be an added benefit (approx 15-20s) to remove intro parts that do not reflect on the true character of the song (spoken intros, soft intros to punchy songs, etc)

Noise and Gain Automation reduce the prominence of the original signal by either applying noise to it (which "blurs out" small range audio features to the human ear) or randomly reducing the gain, or volume, of the track over the course of the signal. Again, these are applied in [36] and [48]

[48] Shows by an ablation study that even without student training, audio augmentations improve the performance of the model by reducing overfitting. Generally, these augmentations are applied stochastically in series (stacked one upon the other), meaning that not all signals going through the pipeline will be subject to the same modifications. We consider using audio augmentations later on in the project when all base models are trained on a non-augmented dataset.