Audio Engineering Society

Convention Express Paper 6

Presented at the 153rd Convention
2022 October

This Express Paper was selected on the basis of a submitted synopsis that has been peer reviewed by at least two qualified anonymous reviewers. The complete manuscript was not peer reviewed. This express paper has been reproduced from the author's advance manuscript without editing, corrections, or consideration by the Review Board. The AES takes no responsibility for the contents. This paper is available in the AES E-Library (http://www.aes.org/e-lib), all rights reserved. Reproduction of this paper, or any portion thereof, is not permitted without direct permission from the Journal of the Audio Engineering Society.

1D Convolutional Layers to Create Frequency-Based Spectral Features for Audio Networks

Elias Nemer and Greg Vines

Irvine, CA, U.S.A.

Correspondence should be addressed to E. Nemer (enemer@ieee.org)

ABSTRACT

Time-Frequency transformation and spectral representations of audio signals are commonly used in various machine learning applications. Training networks on frequency features such as the Mel-Spectrogram or Chromagram have been proven more effective and convenient than training on time samples. In practical realizations, these features are created on a different processor and/or pre-computed and stored on disk, requiring additional efforts and making it difficult to experiment with various combinations. In this paper, we provide a `PyTorch` framework for creating spectral features and time-frequency transformation using the built-in trainable `conv1d()` layer. This allows computing these on-the-fly as part of a larger network and enabling easier experimentation with various parameters. Our work extends the work in the literature developed for that end: First by adding more of these features; and also by allowing the possibility of either training from initialized kernels or training from random values and converging to the desired solution. The code is written as a template of classes and scripts that users may integrate into their own `PyTorch` classes for various applications.

1 Introduction

1.1 Context and Motivation

Time-Frequency representation of audio signals has been used in various audio processing applications, well before neural networks. The justification being that this paradigm allows modelling the peculiarities of the auditory system when it comes to resolution, perception, or masking. Invertible transformations such as the Discrete Fourier Transform (DFT) are desirable in applications where the time signal needs to be re-synthesized following processing. This is the case for applications such as echo cancellation [1] noise reduction [2] or de-reverberation [3] .

In addition to time-frequency transformation, spectral representations are important to capture or visualize the two domains' dynamics when analyzing a long audio segment. Representations such as the Mel-Spectrogram [4], Chromagram [5] or Cochleogram [6] are some examples, used in applications where estimation [7], classification or detection [8] is sought.

Spectral representations have also been used in machine learning (ML), due to their proven relevance in traditional approaches. Moreover, some of the network configurations used in ML-based image processing are being adapted to audio applications by treating spectral features as 2D 'images'. In some applications, up to three different

spectral features may be combined together in a multi-layer 'image' and processed using 2D `conv2d()` layers [9].

In off-line applications spectral features are generated during pre-processing and stored on disk or memory prior to training. In real-time applications utilizing special ML hardware, spectral features are typically generated on a side processor and require significant processing power and memory, as well as the bandwidth and synchronization with the edge processor running the network. It is therefore desirable to generate spectral features on-the-fly, as part of the neural network: First it saves on additional efforts and storage for pre-processing; in addition, it allows easier experimentation with various spectral parameters or with combining spectral features.

The `Pytorch` framework [10] provides a trainable `conv1d()` function meant for 1D signal inputs such as audio sequences. Setting parameters such as the stride, padding and dilation allows flexibility in how the layer can be used.

1.2 Published Work

`Tensorflow` [11] has a `tf.signal` package that performs the Fast Fourier Transform (FFT) and Short-Time Fourier Transform (STFT) on GPUs. `Kapre` [12] – which is part of the `Keras` API, and `torchaudio` –which is part of `PyTorch` support audio to Mel Spectrogram conversion, but only `Kapre` supports signal processing with trainable kernels for time-frequency transformations.

The `nnAudio` toolbox [13] is a comprehensive framework that includes trainable networks for the STFT, Mel-Spectrogram and the Constant-Q transform. The networks are initialized with the analytical kernel expressions and made to continuously train on various audio signals. In [14], we provided a `PyTorch` framework for generating the Cochleogram as well as the time-domain complex filter-banks for analysis and re-synthesis using the built-in trainable `conv1d`() layer.

1.3 Contribution of this Paper

This paper extends some of the work done in the literature, and provides the code that users can integrate in their `PyTorch` classes. In addition to deriving the analytical solutions for the `conv1d()` kernels, we show that it is possible to train a network starting from random kernels and make it converge to the desired solution for the DFT, IDFT, Mel-Spectrogram and Chromagram. This may be desirable for the case where one wants to train on a specific variation of these features that is provided as a 'black-box' implementation by a 3rd party library. We show that achieving such convergence is possible by combining loss functions from various parts of the network as well as using the proper training data and adaptation schemes.

The Github codebase includes the code in the form of `PyTorch` Classes that users can add their own blocks to, or simply integrate into their own networks.

2 Discrete Fourier Transform

2.1 Definition and Notation

The Discrete Fourier Transform (DFT) maps a discrete sequence of real or complex values: $x(n)$ unto a complex sequence $Y(k)$ of the same length. The DFT is a special case of the Z-transform and defined as [15]:

$$Y(k)=\sum_{n=0}^{N-1}x(n)\cdot e^{-j\cdot k\cdot n\cdot\frac{2\pi}{N}} \quad n,k=0,1,.....,N-1 \quad (1)$$

Where n is the index in the incoming sequence and k the index of the transformed sequence. The above definition is applicable to a finite length sequence N and often referred to as the Short-Time Fourier Transform (STFT), as opposed to the case of an infinitely long sequence. The DFT is invertible and the original sequence $x(n)$ can be recovered from the transformed $Y(k)$ via an inverse transformation:

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}Y(k)\cdot e^{j\cdot k\cdot n\cdot\frac{2\pi}{N}} \quad n,k=0,1,.....,N-1 \quad (2)$$

In audio signal processing, the DFT is used to map a sequence of N discrete audio samples in the time domain unto a complex N sequence that represents the frequency content of the signal. This transformation is instrumental for a number of audio algorithms both classical and machine-learning based. Due to the properties of the DFT, the complex transform of a real audio sequence has symmetry around the $N/2$ point, and thus only the first $(N/2+1)$ points are needed in $Y(k)$.

2.2 The DFT as a Convolution

Both the DFT (Eq.1) and its inverse (Eq.2) can be thought of as a convolution or filtering operation and implemented using a bank of filters. The definition of the convolution implemented in the `Conv1d()` is considered to derive the expression of the filters. First Eq.1 is split in its real and imaginary parts:

$$Y(k)\big|_{real} = \sum_{n=0}^{N-1} x(n)\cos\left(k \cdot n \tfrac{2\pi}{N}\right) \quad (3)$$

$$Y(k)\big|_{imag} = -\sum_{n=0}^{N-1} x(n)\sin\left(k \cdot n \tfrac{2\pi}{N}\right) \quad (4)$$

The classical definition of the convolution between an audio sequence $x(n)$ and a time filter $h_k(n)$ is given by:

$$y_conv(n)\big|_k = \sum_{m=0}^{N-1} x(n-m)\, h_k(m) \quad (5)$$

However, in common ML libraries such as `PyTorch`, the `conv1d()` functionality is implemented using a modified version of the classical definition. Namely Eq. 5 becomes (Fig.2):

$$y_conv(n)\big|_k = \sum_{m=n-N+1}^{n} x(m)\, h_k(m) \quad (6)$$

If we were to implement the DFT using a bank of filters, the expression of each filter generating the k^{th} frequency point may be derived from the equations above. By considering a given value of N, and by equating Eq. 3 to Eq. 6, it can be shown that the expressions for the real and imaginary filters are given by:

$$h_{k-real}(n)\big|_{DFT} = \cos\left(n \cdot k \tfrac{2\pi}{N}\right) \quad (7)$$

$$h_{k-imag}(n)\big|_{DFT} = -\sin\left(n \cdot k \tfrac{2\pi}{N}\right) \quad (8)$$

The length of the filters in this case is *N*, with *n* being the time index and *k* the frequency index. Due to the symmetry of the DFT, $k = 0,..,N/2$. In the context of Fig. 2, $x(N-1)$ represents the current input sample, $x(0)$ the oldest sample, $y(N-1)$ the current desired output, and *s* is the 'stride' or frame overlap.

2.3 Network Configuration with Pytorch

2.3.1 Number and Length of Kernels

To implement the DFT using the `conv1d()`, we need $(N/2+1)$ *real* kernels and *imaginary* kernels, as shown in Fig. 1. The kernels may be initialized with the analytical solution (Eq.7, Eq.8) if desired, or simply started with random values.

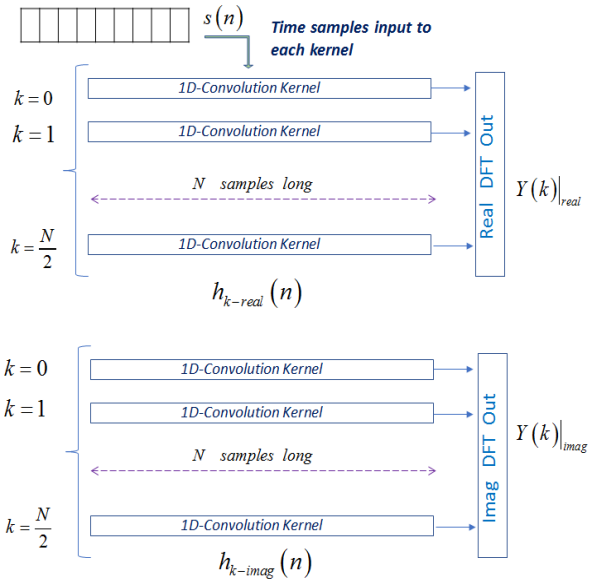


Figure 1: Real and Imaginary DFT Kernels

2.3.2 Kernel length and Stride

In the `conv1d()` layer, when the input is of size *N* and the kernel is of length *N*, only one output is generated at each of the kernels. This is what's desired when doing a DFT of an *N*-point sequence. When the input is longer than *N*, the number of outputs can be determined by setting the proper stride in the `conv1d()` kernel. This is equivalent to the overlap when implementing a frame-based DFT. Consider a 50% overlap, or simply a stride of *N/2*, the output for a long input sequence is shown in Fig. 2, for the simple example of *N=4*.

2.4 Inverse DFT

2.4.1 Definition

In a similar way, a set of convolutional banks may be used to compute the inverse DFT and recover the original sequence $x(n)$. First, Eq. 2 is expanded in its real and imaginary parts (ignoring the normalizing factor for now) as:

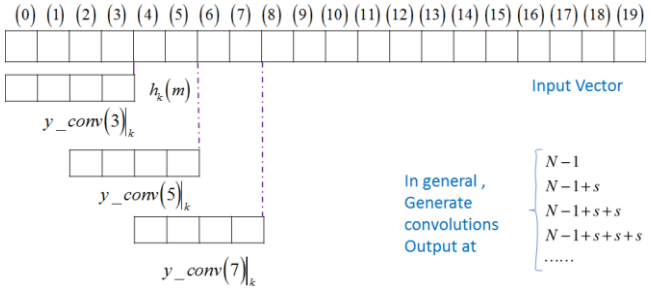


Figure 2 : Outputs for Stride = 2 and N = 4

$$x(n)\big|_{re} = \sum_{k=0}^{N-1} Y(k)\big|_{re} \cos\left(kn\tfrac{2\pi}{N}\right) - \sum_{k=0}^{N-1} Y(k)\big|_{im} \sin\left(kn\tfrac{2\pi}{N}\right) \quad (9)$$

$$x(n)\big|_{im} = \sum_{k=0}^{N-1} Y(k)\big|_{re} \sin\left(kn\tfrac{2\pi}{N}\right) + \sum_{k=0}^{N-1} Y(k)\big|_{im} \cos\left(kn\tfrac{2\pi}{N}\right) \quad (10)$$

For an audio signal, the time domain sequence $x(n)$ is real, and the imaginary part (Eq. 10) is ignored. Moreover, since the real $Y(k)\big|_{real}$ and imaginary $Y(k)\big|_{imag}$ parts of the DFT only span half the points, Eq 9 is re-written with the corrected summation limits as:

$$x(n)\big|_{re} = \sum_{k=0}^{N/2+1} Y(k)\big|_{re} \cos\left(kn\tfrac{2\pi}{N}\right) - \sum_{k=0}^{N/2+1} Y(k)\big|_{im} \sin\left(kn\tfrac{2\pi}{N}\right) \quad (11)$$

2.4.2 Kernel Configuration

By repeating the analysis done in the previous sections, we can derive the expressions for the kernels. Namely, the network consists of two convolution banks, each having N kernels of length $(N/2+1)$, and with the following expressions:

$$h_{k-real}(k)\big|_{IDFT} = \cos\left(n \cdot k\tfrac{2\pi}{N}\right) \quad (12)$$

$$h_{k-imag}(k)\big|_{IDFT} = \sin\left(n \cdot k\tfrac{2\pi}{N}\right) \quad (13)$$

Here, n is the output index and relates to the kernel number with its span: $n = 0,\ldots,N-1$, whereas k is the sample index and relates to the kernel length with its span being $k = 0,\ldots,N/2+1$. The operation is illustrated in Fig. 3. Note that the normalizing factor $1/(N/2+1)$ has been omitted in Eq. 12 and Eq. 13 and can be absorbed as a scaling factor in the final results. The DFT loss is the mean squared error (MSE) between the output DFT vectors of the model and the desired DFT (computed from any common library).

$$\underset{DFT}{loss} = \sum_{k=0}^{N/2+1} \left(Y(k)\big|_{re} - L(k)\big|_{re}\right)^2 + \left(Y(k)\big|_{im} - L(k)\big|_{im}\right)^2 \quad (14)$$

Similarly, the IDFT loss is the smooth L1 norm between the input and the synthesized time signals.

$$loss_{IDFT} = \sum_{k=0}^{N-1} \left|x(n) - s(n)\right|^P \quad (15)$$

With P changing from 1 to 2 as the loss goes below 1. Various data sources were tried, such as audio recording, chirps, AM-modulated signal and various noises. Experimentation showed that using white Gaussian noise for training is the most effective alternative. This is likely because white noise stimulates all frequency points equally, and does not overemphasize any one kernel during adaptation.

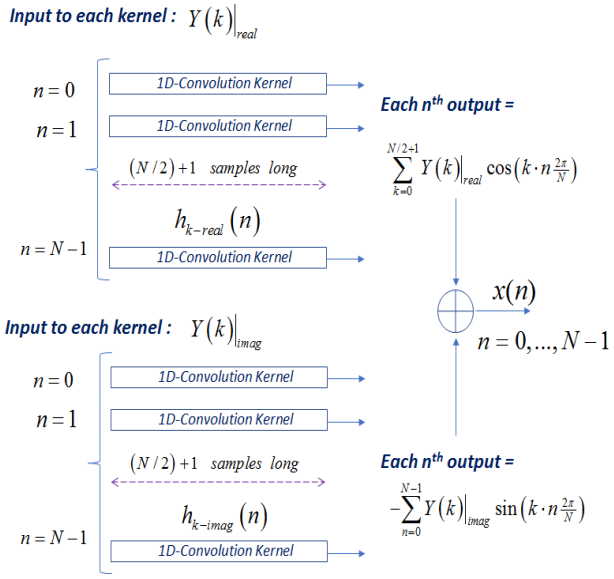


Figure 3: Kernels to Generate the Inverse DFT

The parameters for the Pytorch `conv1d()` layers are shown in Table 1 below.

Kernel	In_channels	Out_channels	Kernel size	stride
DFT real	1	N/2 + 1	N	N/2
DFT imag	1	N/2 + 1	N	N/2
iDFT real	1	N	N/2 + 1	N/2

Table 1 : Pytorch `conv1d()` parameters

2.5 Training

The model is trained one frame at a time. The loss function computed on each frame is used to back-propagate and update the weights. The training is done sequentially: The DFT kernels are trained first and once convergence is achieved, they are frozen and the IDFT kernels are trained separately. Different optimization schemes were used and it was found that the Stochastic Gradient Descent (SGD) gives the best convergence.

Two training scenarios are considered: 1) start with random kernels or 2) initialize the kernels for the DFT and IDFT using Eq. 7, 8, 12, and 13. In both cases the model is trained to full convergence and the initial and final kernels are plotted and compared.

2.5.1 From Random Kernels

The DFT kernels are adapted using an (SGD) optimizer, for 200 epochs, followed by the adaptation of the inverse DFT kernels. The training error is shown in Fig. 4, and converges to the order of $1e-5$ for each of the two sections.

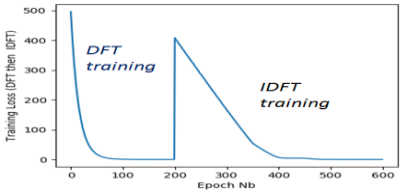


Figure 4: DFT and IDFT Training Loss

Figure 5 shows the initial and final values of a subset of the DFT kernels. On close examination, the solution converges to the analytical solution derived earlier (Eq. 7 and 8) to a mean squared error (MSE) of the order of 7e-7.

2.5.2 From Initialized Kernels

When the initial kernels are loaded, the re-synthesized signal is verified to be equal to the original input –within an MSE of $1e-6$, before any further training. If the same training used above is carried out, we find the initial and final kernels to be identical for the DFT as well as the inverse DFT kernels (to within an MSE of $1e-10$). Therefore training the model with random data from initialized kernels does not cause a deviation from the initial, i.e. the analytical solution.

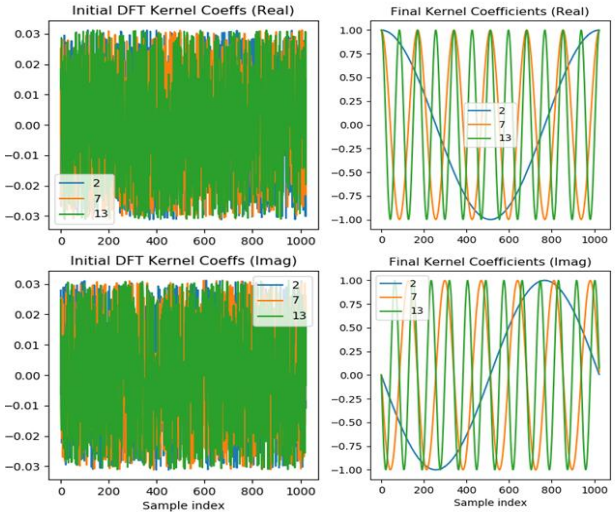


Figure 5: Initial (L) and Final (R) DFT Kernels

3 Mel-Spectrogram

3.1 Definition

A Spectrogram is a convenient way to visualize the squared magnitude of the DFT points for each analysis frame. It is a 3D plot with the *x* and *y* axis being the time and frequency indices respectively. The third dimension is often replaced with an intensity value to view the plot in 2D. The *Mel* scale came about with the realization that human hearing does not perceive frequency on a linear, but a more logarithmic-like, scale [16]. The *Mel* scale was developed based on having a unit of pitch such that equal distances in pitch sounded equally distant to the listener. A transformation from frequency in *Hz* to the *Mel* pitch scale is given by [17]:

$$m_{mel}(f) = 2595\log_{10}\left(1+\frac{f}{700}\right) \qquad (16)$$

With the reverse transformation from *Mel* to *Hz* as:

$$f_{Hz}(m) = 700\left(10^{m/2595}-1\right) \qquad (17)$$

3.2 Functional Block Diagram

The *Mel*-Spectrogram is generated by first computing a DFT on the signal, using a given frame length (*N*) and an overlap. We then compute the so-

called *Mel*-scale filterbank, which is a set of M filters that are applied to the spectrogram – or squared DFT magnitudes. Each one of these filters combines the energy of a number of frequency points representing the *Mel* unit. Each vector representing the filter is mostly zero except for a small segment of the spectrum.

3.3 Determining the Mel Filters

The span and values of each filter coefficients are a function of the sampling frequency, the number of DFT points and the desired bandwidth of the signal to cover. The key steps are as follow:

- Define the frequency range and compute the end points in *Hz* and *Mels*.
- Determine the number of filters, and define equally spaced frequencies in *Mels* using Eq. 16.
- Convert these frequencies to Hz using Eq. 17.
- Translate these frequencies to bins, based on the STFT size and generate the set $f(m)$ in bins for $m = 1,...,M$.
- Construct the *M* filters: $H_m(k) \quad m = 1,...,M$

$$H_m(k) = \begin{cases} 0 & \cdots\cdots for \cdots\cdots \quad k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \\ 1 & k = f(m) \end{cases}$$

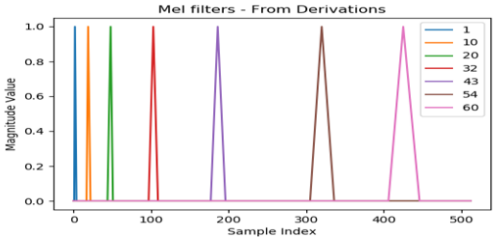


Figure 6: Selected Mel Filters

Figure 6 illustrates 7 of 64 *Mel* filters for the case of an STFT of 512 bins and a sampling rate of 22kHz. Note that the `Librosa` library contains a Melgram filter function, but produces a slightly different set, partly due to special normalization used.

3.4 Convolutional Model

The *Mel* filters can be implemented using a bank of convolutional layer `conv1d()`. The number of kernels, or output dimension, is the number of desired *Mel* filters, for instance 32. The length of each kernel is simply the number of frequency points coming out of the DFT. The input dimension is one, since the same DFT vector goes to all the kernels. Finally the stride is not relevant and may be set to any value. The full network consists of the real and imaginary DFT kernels from the previous sections along with M kernels of length $N/2+1$ each, as shown in Fig. 7 below.

3.4.1 Loss Functions, Data, Optimizer

One key result of this work is that using the output of the *Mel* power values (Fig.7) alone is not enough to converge the network to a good solution. Instead we found that combining the loss from the DFT outputs along with the *Mel* output yields good results. Therefore two loss variables are created:

$$\underset{DFT}{Loss} = \sum_{k=0}^{N/2+1} \left| Y(k)\big|_{re} - L(k)\big|_{re} \right|^P + \left| Y(k)\big|_{im} - L(k)\big|_{im} \right|^P \quad (18)$$

Where L refers to the Label, in this case the library FFT, and:

$$\underset{Mel}{Loss} = \sum_{m=0}^{n_mels-1} \left| out(m)\big|_{NN} - out(m)\big|_{librosa} \right|^P \quad (19)$$

With P as in Eq. 15. The sum of the two loss function is then used to back-propagate and update the weights.

$$TotalLoss = DFT\ Loss + Mel\ Loss \quad (20)$$

As before, random data is generated in batches with each consisting of White Gaussian noise. Using the SGD optimizer (as in the case of the DFT) requires varying the learning rate in order to converge deep enough. It was found that using the `Adagrad` optimizer [18] provides a smoother and deeper convergence without making continuous changes to the parameters. It is likely because the `Adagrad` optimizer is implicitly changing the effective learning rate in each dimension of the gradient, based on the gradient energy.

3.5 Model Training

As before, two scenarios are considered: 1) Initialize the kernels with the analytical solution or 2) start from random.

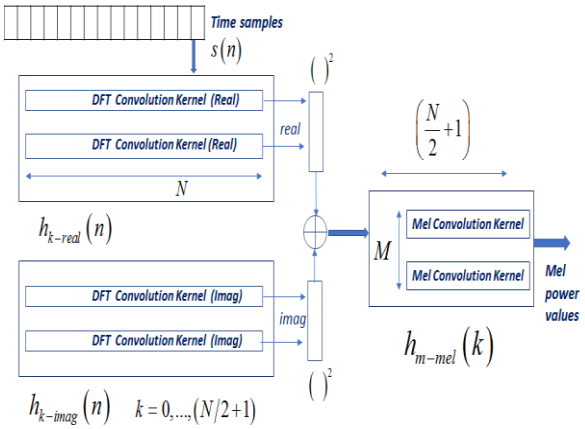


Figure 7: Mel Network Configuration

3.5.1 *Training from Random Kernels*

In this mode, the kernels are not initialized, and the total loss is used to backpropagate and converge. Figure 8 shows the training loss (in log scale) over 600 epochs.

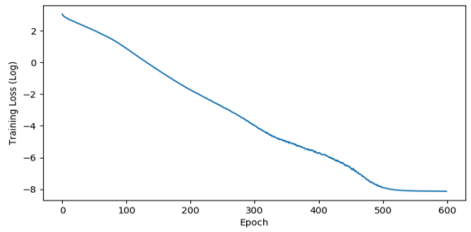


Figure 8: Total Training Loss (Eq. 20)

Figure 9 shows the initial and final kernels of the *Mel* filters in an intensity-based 3D plot. Upon close examination, these do converge to the analytical solution (Fig 6): the MSE between the analytical and final kernels is of the order of $4e-8$.

3.5.2 *Training from Initialized Kernels*

When training from initialized kernels, the same combined loss is used as before. The initial and final kernels are examined, as well as the model is evaluated using an entire audio utterance.

It was found that the initial and final *Mel* filters are identical: The actual computed MSE between them is of the order of $2e-11$. Thus the model does not deviate from the initial analytical solution when white Gaussian noise is used.

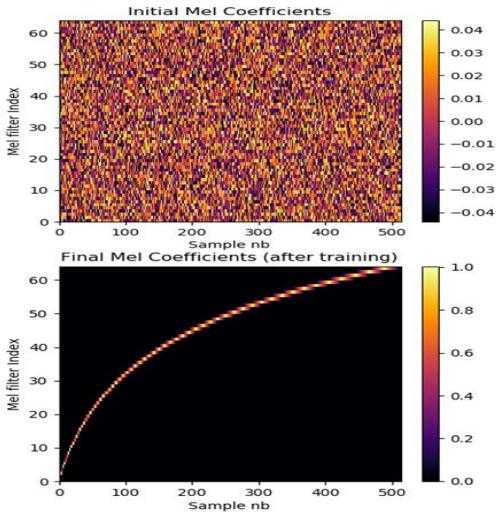


Figure 9: Initial (T) and Final (B) Mel Kernels

4 Chromagram

4.1 Definition

In music analysis, pitch is a perceptual property of sounds and allows their ordering on a frequency-related scale. It is the quality that makes it possible to judge sounds as "higher" or "lower" in pitch. The human perception of pitch is periodic in the sense that two pitches are perceived as similar in “color”, or harmonic role, if they differ by one or several octaves. Based on this observation, a pitch can be separated into two components, which are referred to as tone height and Chroma, and sometimes called the pitch helix [19]. The Chroma, or color, captures the harmonic and melodic characteristics of music, and is robust to changes in timbre and instrumentation. Such features are commonly used in music analysis, where the pitches can be meaningfully categorized. Assuming the equal-tempered scale, one considers twelve Chroma values represented by the set {C, C♯, D, D♯, E , F, F♯, G, G♯, A, A♯, B} which are the twelve pitch spelling attributes used in Western music notation.

From a signal processing perspective, the term Chroma vector, or Chromagram, relates to the twelve pitch classes. It is typically a 12-element feature vector, though it may contain more elements,

and indicates how much energy from each pitch class {C, C#,D, D#, E, ..., B} is present in the signal.

4.2 Functional Implementation

4.2.1 Using DFT

The conversion of an audio recording into a Chroma representation, or Chromagram, may be performed by using the Short-Time Fourier Transform (STFT) in combination with binning strategies [20]. The idea is to aggregate all spectral information that relates to a given pitch class into a single coefficient.

4.2.2 Determining the Chroma Filters

The Chroma filters are somewhat similar to the Mel filters described earlier. They combine the energies of a selected set of frequency bins to yield the desired filter output. In this case each Chroma filter combines the energy of one pitch class, for example the C# pitch class, from the various bins of the DFT vector which conceptually represent the various octaves.

In this work, we make use of the `Librosa` library function to synthesize such filters, given the number of DFT points, sampling rate, and the number of filters. An example is given in Fig. 10 below showing 4 of 24 filters.

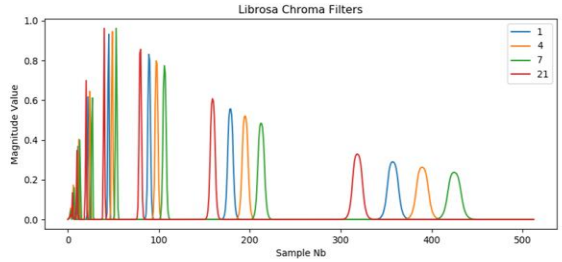


Figure 10 : Subset of Chroma Filters (From Librosa)

4.3 Convolutional Model

The network configuration is similar to the one used in the Mel Spectrogram. Namely, the Chroma filters are implemented with a bank of `conv1d()` layer. The number of kernels, or output dimension, is the number of desired Chroma filters -e.g. 24. The length of each kernel is the number of frequency points coming out of the DFT -e.g. 512. The input dimension is simply one, because the same DFT vector goes to all the kernels. Finally, the stride is not relevant and is set to any value.

The full network consists of the real and imaginary DFT kernels from the previous sections along with the C Chroma kernels of length $N/2+1$ each, as shown in Fig. 11.

4.3.1 Loss Function and Optimizers

We made a similar observation here as in the case of the Mel-spectrogram. Namely, in order to get the network to converge for the case where we start with random kernels, two loss functions are combined to yield a total loss. The two loss variables (smooth L1 norm) are summed and the total loss is used in the back-propagation.

$$\underset{DFT}{Loss} = \sum_{k=0}^{N/2+1} \left| Y(k)\big|_{re} - L(k)\big|_{re} \right|^{P} + \left| Y(k)\big|_{im} - L(k)\big|_{im} \right|^{P} \quad (21)$$

Where L refers to the Label, in this case the library FFT, and:

$$\underset{Chroma}{Loss} = \sum_{m=0}^{n_chroma-1} \left| out(m)\big|_{NN} - out(m)\big|_{librosa} \right|^{P} \quad (22)$$

$$TotalLoss = DFT\ Loss + Chroma\ Loss \quad (23)$$

Where P changes from 1 to 2, as the Loss function goes below 1. As before, random data is generated in batches with each consisting of White Gaussian noise. Similarly, it was found that using the `Adagrad` optimizer yields a smooth and deep convergence trajectory without making adjustments to the learning rate.

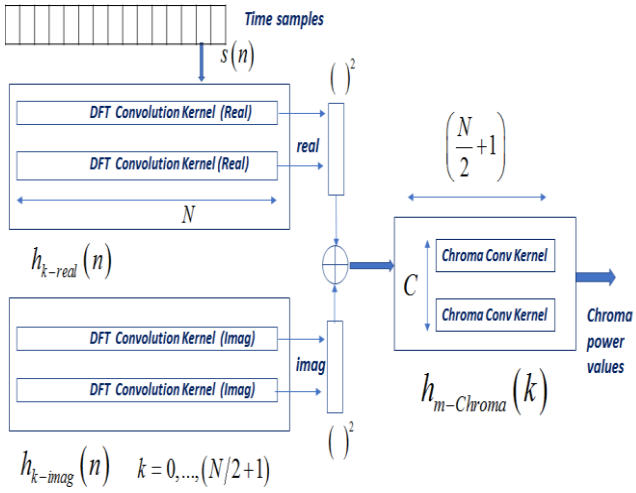


Figure 11: Chromagram Network Configuration

4.4 Training

The model is trained one frame at a time. As before, two scenarios are considered: 1) Initialize the kernels with the analytical solution or 2) start from random kernels.

4.4.1 Training from Random Kernels

In this mode all kernels start with random values, and adaptation proceeds, in this example, for 700 epochs. The total loss (in log scale) is plotted in Fig. 12.

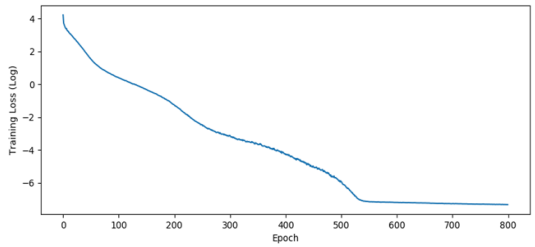


Figure 12: Total Training Loss (Eq. 23)

Figure 13 shows the initial and final values of the Chroma kernels after convergence. As before, we find that these converge to the analytical solution within an MSE of $7e-8$.

4.4.2 Training from Initialized Kernels

When training from initialized kernels, the same combined loss is used as before. The initial and final kernels are examined and found to be identical (to within an MSE of $4e-10$); thus the model does not deviate from the initial analytical solution.

5 Github

The Github open source code is found in [21]. The code consists of various folders, each referring to one of the models described in this paper. In each folder, a readme file explains how to run the model and what parameters can be changed.

6 Conclusion

This paper presented a number of 1D convolutional networks for the realization of commonly used spectral features in ML audio applications. These networks are based on the `Pytorch conv1d()` layer and may be integrated in a larger network containing additional application-specific layers. The spectral features addressed consist of the DFT, its inverse, the Mel-Spectrogram, and the Chromagram.

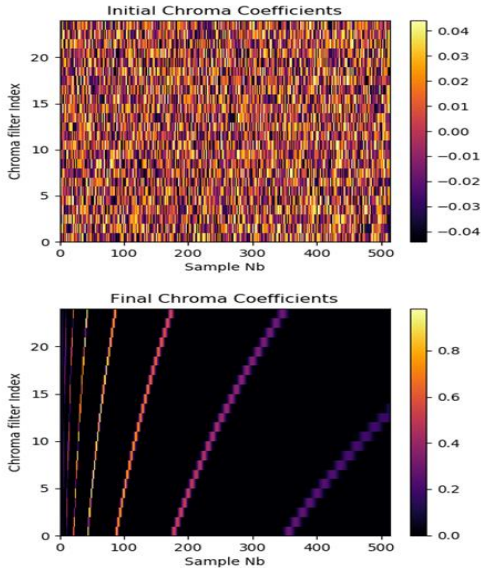


Figure 13: Initial and Final Chroma Kernels

We derived the analytical expressions for the `conv1d()` kernels and demonstrated that the network may be initialized from the analytical solution or simply trained from random values and made to converge to these desired solutions. The objective was to show that one can configure and train a simple network to replicate the functionality of a 3rd party realization of these spectral features, be it a library function or some modified implementation that one has access to. The training is done in a black-box approach in that we only need the desired output for one frame at a time in order to reproduce the functionality of that feature. We showed that the networks consist of simple 1 or 2 `conv1d()` layers and quadratic non-linearity. Arguably, if a more complicated form of these features is to be implemented, more layers may be needed. The networks realized here serve as a base and may be extended to include more layers.

The ability to generate spectral features on-the-fly as part of a larger neural network is desirable. First, it saves on additional efforts and storage for data preprocessing. In addition, it allows easier

experimentation with various spectral parameters or various combinations of spectral features.

7 References

[1] J.M. Borrallo, M. G. Otero, “On the Implementation of a Partitioned Block Frequency Domain Adaptive Filter (PBFDAF) for Long Acoustic echo cancellation”. *Signal Processing*, vol. 27, issue 3, 1992, pp. 301-315, ISSN 0165-1684.

[2] S. F. Boll, “Suppression of Acoustic Noise in Speech Using Spectral Subtraction”, *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 27, no. 2, pp. 113–120, 1979.

[3] K. Lebart, J. M. Boucher. “A New Method Based on Spectral Subtraction for Speech De-reverberation”, *ACUSTICA* Vol. 7 pp. 359 – 366. 2001.

[4] K. Prahallad. “Spectrogram, Cepstrum and Mel-Frequency Analysis”. Available online at : http://www.speech.cs.cmu.edu/15492/slides/03_mfcc.pdf.

[5] N. Lenssen and D. Needell. "An Introduction to Fourier Analysis with Applications to Music", *Journal of Humanistic Mathematics*, vol. 4, issue 1, pp. 72-91, 2014. DOI: 10.5642/jhummath.201401.05. Available online at: https://scholarship.claremont.edu/jhm/vol4/iss1/5 .

[6] R. F. Lyon, "A Computational Model of Filtering, Detection, and Compression in the Cochlea", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1282-1285. (1982).

[7] R. Martin, “Noise Power Spectral Density Estimation Based on Optimal Smoothing and Minimum Statistics,” *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 5, pp. 504–512, 2001.

[8] B. Gao, W. Woo, L.C. Khor. “Cochleagram-based Audio Pattern Separation Using Two-dimensional Non-Negative Matrix Factorization with Automatic Sparsity Adaptation”. *Journal of Acoustical Society of America*. 2014 Mar;135(3):1171-85. doi: 10.1121/1.4864294. PMID: 24606260.

[9] I. McLoughlin, Z. Xie, Y. Song. “Time–Frequency Feature Fusion for Noise Robust Audio Event Classification”. *Circuits Systems Signal Processing* 39, 1672–1687 (2020). https://doi.org/10.1007/s00034-019-01203-0.

[10] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In *H. Wallach et al., eds.* Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. 2019 Available online at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[11] M. Abadi *et al*. “Tensorflow: Large-Scale Machine Learning on Heterogeneous Systems”. 2015. Available online at: http://tensorow.org/

[12] K. Choi, D. Joo, and J. Kim, “Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep Neural Network Models with Keras”. 2017.

[13] K. W. Cheuk, H. Anderson, K. Agres and D. Herremans, "nnAudio: An on-the-Fly GPU Audio to Spectrogram Conversion Toolbox Using 1D Convolutional Neural Networks", in *IEEE Access*, vol. 8, pp. 161981-162003, 2020, doi: 10.1109/ACCESS.2020.3019084.

[14] E. Nemer, "Audio Cochleogram with Analysis and Synthesis Banks Using 1D Convolutional Networks", in 1st *International Conference on Artificial Intelligence and Data Analytics (CAIDA)*, 2021, doi: 10.1109/CAIDA51941.2021.9425342.

[15] S. W. Smith. “The Scientist and Engineer's Guide to Digital Signal Processing”, chapter 8; online at : https://www.dspguide.com/ch8.htm.

[16] S. Umesh, L. Cohen, and D. Nelson, “Fitting the Mel Scale," in Proc. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, Mar. 1999, pp. 217-220.

[17] D. O'Shaughnessy, Speech Communications: Human and Machine. New York, NY, USA: Wiley, 1987.

[18] R. Gylberth “Introduction to Adagrad”. Online: https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827.

[19] M. A. Bartsch and G. H. Wakefield, "Audio Thumbnailing of Popular Music Using Chroma-based Representations", in *IEEE Transactions on Multimedia*, vol. 7, no. 1, pp. 96-104, Feb. 2005, doi: 10.1109/TMM.2004.840597.

[20] E. Gómez, Emilia. "Tonal Description of Music Audio Signals". *Ph.D. Thesis*, UPF Barcelona, Spain. 2006.

[21] E. Nemer, G. Vines. Github repository : https://github.com/CptNemoy/ConvAudio.