INTUITIVE AUDIO DEEP LEARNING SERIES

Audio Deep Learning Made Simple (Part 2): Why Mel Spectrograms perform better

A Gentle Guide to processing audio in Python. What are Mel Spectrograms and how to generate them, in Plain English.

Ketan Doshi · Follow
Published in Towards Data Science
8 min read · Feb 19, 2021

Photo by Jordan on Unsplash

This is the second article in my series on audio deep learning. Now that we know how sound is represented digitally, and that we need to convert it into a spectrogram for use in deep learning architectures, let us understand in more detail how that is done and how we can tune that conversion to get better performance.

Since data preparation is so critical, particularly in the case of audio deep learning models, that will be the focus of the next two articles.

Here's a quick summary of the articles I am planning in the series. My goal throughout will be to understand not just how something works but why it works that way.

1. State-of-the-Art Techniques *(What is sound and how it is digitized. What problems is audio deep learning solving in our daily lives. What are Spectrograms and why they are all-important.)*
2. **Why Mel Spectrograms perform better — this article** *(Processing audio data in Python. What are Mel Spectrograms and how to generate them)*
3. Feature Optimization and Augmentation *(Enhance Spectrograms features for optimal performance by hyper-parameter tuning and data augmentation)*
4. Sound Classification *(End-to-end example and architecture to classify ordinary sounds. Foundational application for a range of scenarios.)*
5. Automatic Speech Recognition *(Speech-to-Text algorithm and architecture, using CTC Loss and Decoding for aligning sequences.)*
6. Beam Search *(Algorithm commonly used by Speech-to-Text and NLP applications to enhance predictions)*

Audio File Formats and Python Libraries

Audio data for your deep learning models will usually start out as digital audio files. From listening to sound recordings and music, we all know that these files are stored in a variety of formats based on how the sound is compressed. Examples of these formats are .wav, .mp3, .wma, .aac, .flac and many more.

Python has some great libraries for audio processing. Librosa is one of the most popular and has an extensive set of features. scipy is also commonly used. If you are

using Pytorch, it has a companion library called torchaudio that is tightly integrated with Pytorch. It doesn't have as much functionality as Librosa, but it is built specifically for deep learning.

They all let you read audio files in different formats. The first step is to load the file. With librosa:

```
import librosa

# Load the audio file
AUDIO_FILE = './audio.wav'
samples, sample_rate = librosa.load(AUDIO_FILE, sr=None)
```

load_audio.py hosted with ❤ by GitHub view raw

Or, you can also do the same thing using scipy:

```
from scipy.io import wavfile
sample_rate, samples = wavfile.read(AUDIO_FILE)
```

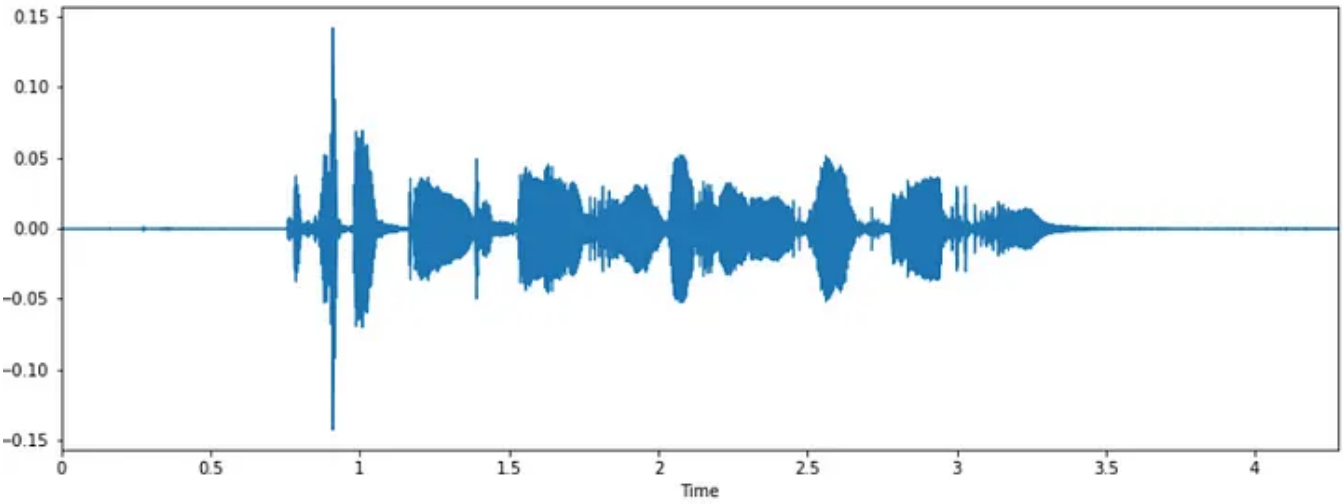
load_audio_scipy.py hosted with ❤ by GitHub view raw

You can then visualize the sound wave:

```
import librosa.display
import matplotlib.pyplot as plt

# x-axis has been converted to time using our sample rate.
# matplotlib plt.plot(y), would output the same figure, but with sample
# number on the x-axis instead of seconds
plt.figure(figsize=(14, 5))
librosa.display.waveplot(samples, sr=sample_rate)
```

display_audio.py hosted with ❤ by GitHub view raw



Visualize the sound wave (Image by Author)

And listen to it. If you are using a Jupyter notebook, you can play the audio directly in a cell.

```
1  from IPython.display import Audio
2  Audio(AUDIO_FILE)
```

play_audio.py hosted with ❤️ by GitHub view raw

Play audio in a notebook cell (Image by Author)

Audio Signal Data

As we saw in the previous article, audio data is obtained by sampling the sound wave at regular time intervals and measuring the intensity or amplitude of the wave at each sample. The metadata for that audio tells us the sampling rate which is the number of samples per second.

When that audio is saved in a file it is in a compressed format. When the file is loaded, it is decompressed and converted into a Numpy array. This array looks the same no matter which file format you started with.

In memory, audio is represented as a time series of numbers, representing the amplitude at each timestep. For instance, if the sample rate was 16800, a one-second clip of audio would have 16800 numbers. Since the measurements are taken at fixed intervals of time, the data contains only the amplitude numbers and not the time

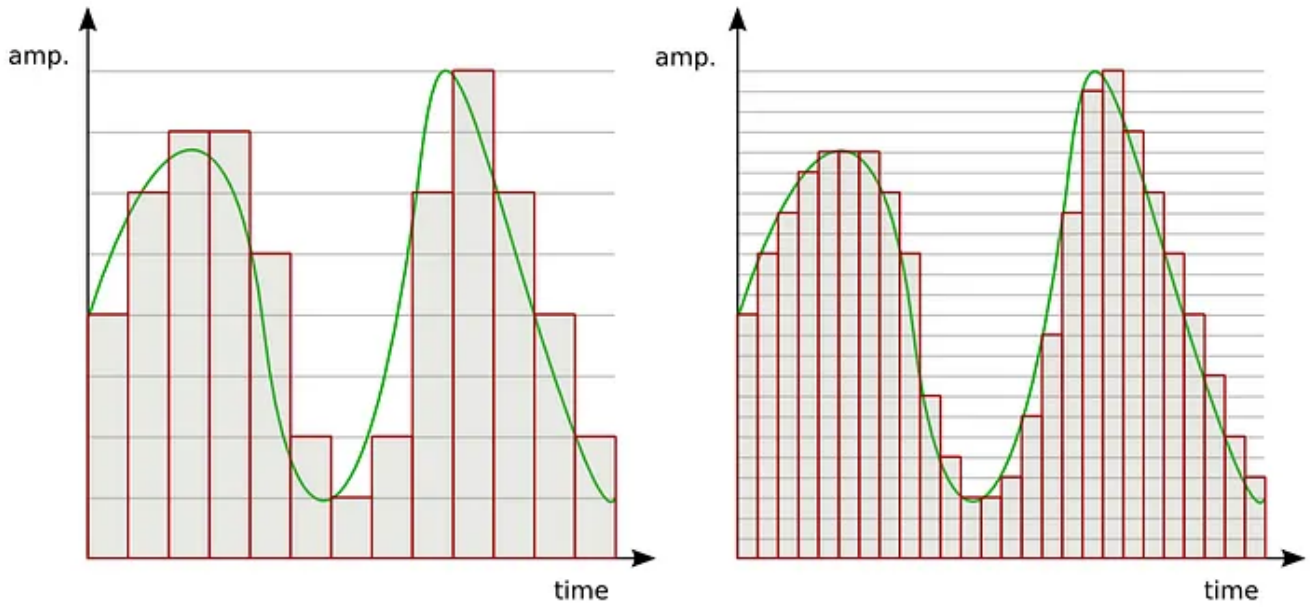
values. Given the sample rate, we can figure out at what time instant each amplitude number measurement was taken.

```
print ('Example shape ', samples.shape, 'Sample rate ', sample_rate, 'Data type', type(s
print (samples[22400:22420])
```

audio_array.py hosted with ❤️ by GitHub view raw

```
Example shape  (68480,) Sample rate  16000 Data type <class 'numpy.ndarray'>
[-0.01196289 -0.00891113 -0.00384521 -0.01159668 -0.0083313  -0.00588989
 -0.01095581 -0.01022339 -0.00222778 -0.012146   -0.00842285 -0.00061035
 -0.0098877  -0.01062012 -0.00445557 -0.0043335  -0.01052856  0.00048828
 -0.00537109 -0.00805664]
```

The bit-depth tells us how many possible values those amplitude measurements for each sample can take. For example, a bit-depth of 16 means that the amplitude number can be between 0 and 65535 ($2^{16} - 1$). The bit-depth influences the resolution of the audio measurement — the higher the bit-depth, the better the audio fidelity.



Bit-depth and sample-rate determine the audio resolution (Source)

Spectrograms

Deep learning models rarely take this raw audio directly as input. As we learned in Part 1, the common practice is to convert the audio into a spectrogram. The spectrogram is a concise 'snapshot' of an audio wave and since it is an image, it is

well suited to being input to CNN-based architectures developed for handling images.

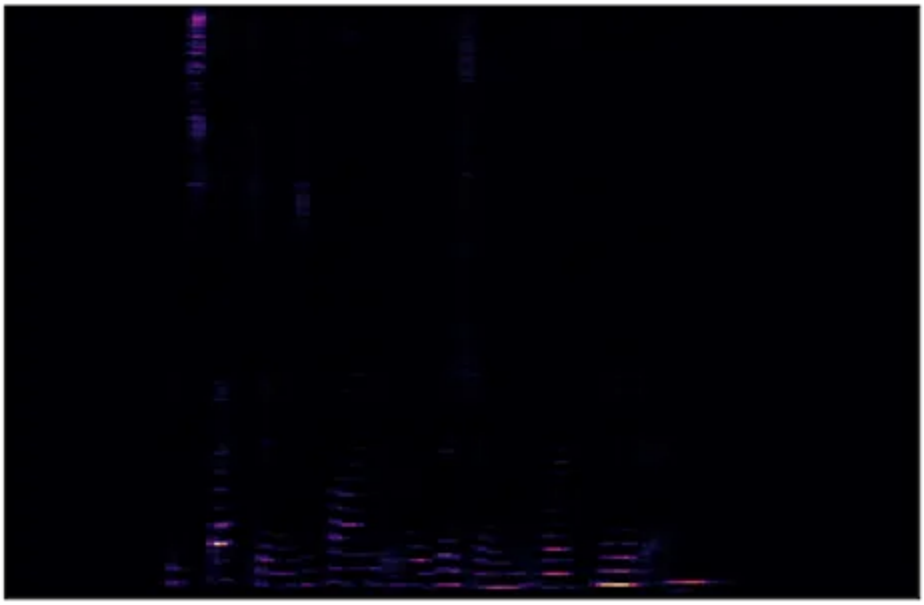
Spectrograms are generated from sound signals using Fourier Transforms. A Fourier Transform decomposes the signal into its constituent frequencies and displays the amplitude of each frequency present in the signal.

A Spectrogram chops up the duration of the sound signal into smaller time segments and then applies the Fourier Transform to each segment, to determine the frequencies contained in that segment. It then combines the Fourier Transforms for all those segments into a single plot.

It plots Frequency (y-axis) vs Time (x-axis) and uses different colors to indicate the Amplitude of each frequency. The brighter the color the higher the energy of the signal.

```
1   sgram = librosa.stft(samples)
2   librosa.display.specshow(sgram)
```

spectrogram.py hosted with ❤️ by GitHub view raw



Simple Spectrogram (Image by Author)

Unfortunately, when we display this spectrogram there isn't much information for us to see. What happened to all those colorful spectrograms we used to see in Science class?

This happens because of the way humans perceive sound. Most of what we are able to hear are concentrated in a narrow range of frequencies and amplitudes. Let's explore that first so we can figure out how to produce those lovely spectrograms.

How do humans hear frequencies?

The way we hear frequencies in sound is known as 'pitch'. It is a subjective impression of the frequency. So a high-pitched sound has a higher frequency than a low-pitched sound. Humans do not perceive frequencies linearly. We are more sensitive to differences between lower frequencies than higher frequencies.

For instance, if you listened to different pairs of sound as follows:

- 100Hz and 200Hz
- 1000Hz and 1100Hz
- 10000Hz and 10100 Hz

What is your *perception* of the "distance" between each pair of sounds? Are you able to tell each pair of sounds apart?

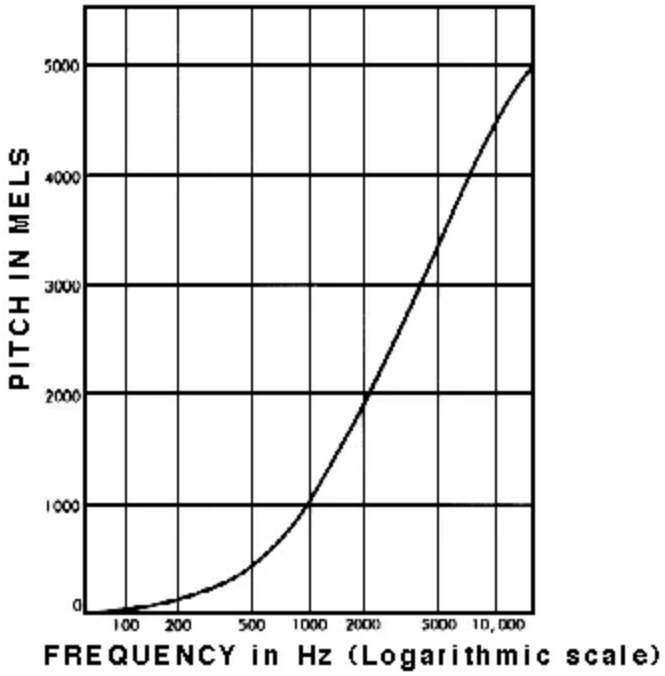
Even though in all cases, the actual frequency difference between each pair is exactly the same at 100 Hz, the pair at 100Hz and 200Hz will *sound further apart* than the pair at 1000Hz and 1100Hz. And you will hardly be able to distinguish between the pair at 10000Hz and 10100Hz.

However, this may seem less surprising if we realize that the 200Hz frequency is actually double the 100Hz, whereas the 10100Hz frequency is only 1% higher than the 10000Hz frequency.

This is how humans perceive frequencies. We hear them on a logarithmic scale rather than a linear scale. How do we account for this in our data?

Mel Scale

The Mel Scale was developed to take this into account by conducting experiments with a large number of listeners. It is a scale of pitches, such that each unit is judged by listeners to be equal in pitch distance from the next.



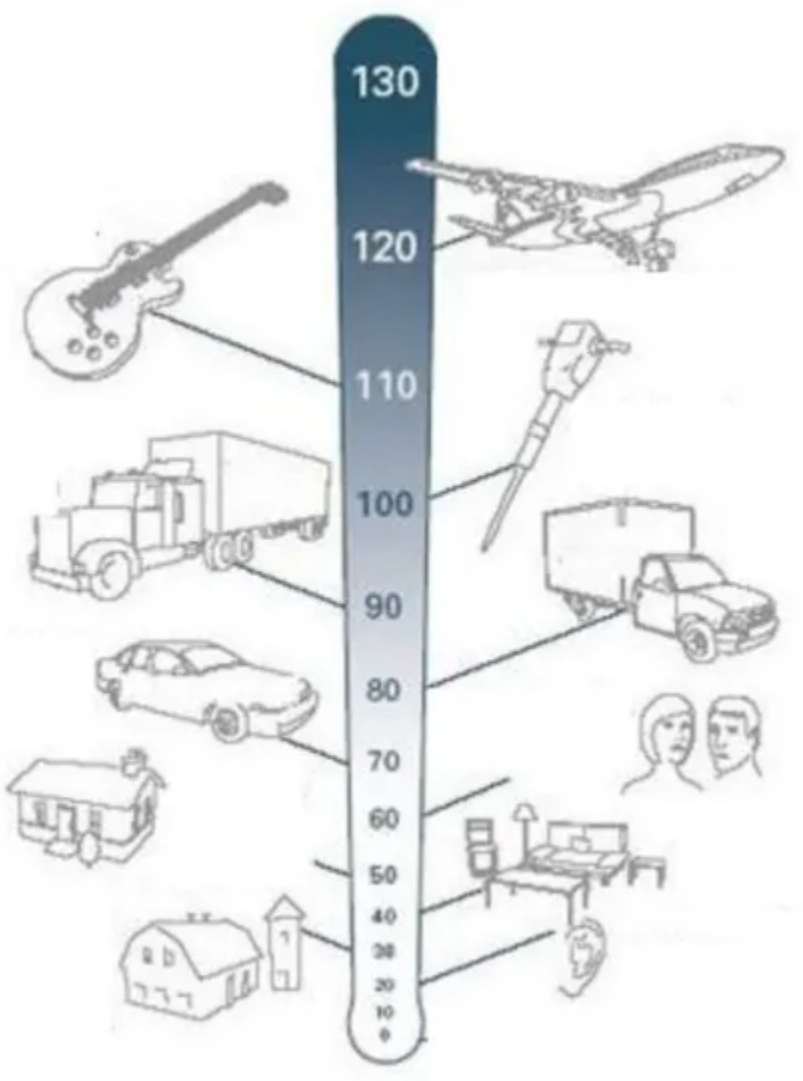
Mel Scale measures human perception of pitch (Source, by permission of Prof Barry Truax)

How do humans hear amplitudes?

The human perception of the amplitude of a sound is its loudness. And similar to frequency, we hear loudness logarithmically rather than linearly. We account for this using the Decibel scale.

Decibel Scale

On this scale, 0 dB is total silence. From there, measurement units increase exponentially. 10 dB is 10 times louder than 0 dB, 20 dB is 100 times louder and 30 dB is 1000 times louder. On this scale, a sound above 100 dB starts to become unbearably loud.



Decibel levels of common sounds (Adapted from Source)

We can see that, to deal with sound in a realistic manner, it is important for us to use a logarithmic scale via the Mel Scale and the Decibel Scale when dealing with Frequencies and Amplitudes in our data.

That is exactly what the Mel Spectrogram is intended to do.

Mel Spectrograms

A Mel Spectrogram makes two important changes relative to a regular Spectrogram that plots Frequency vs Time.

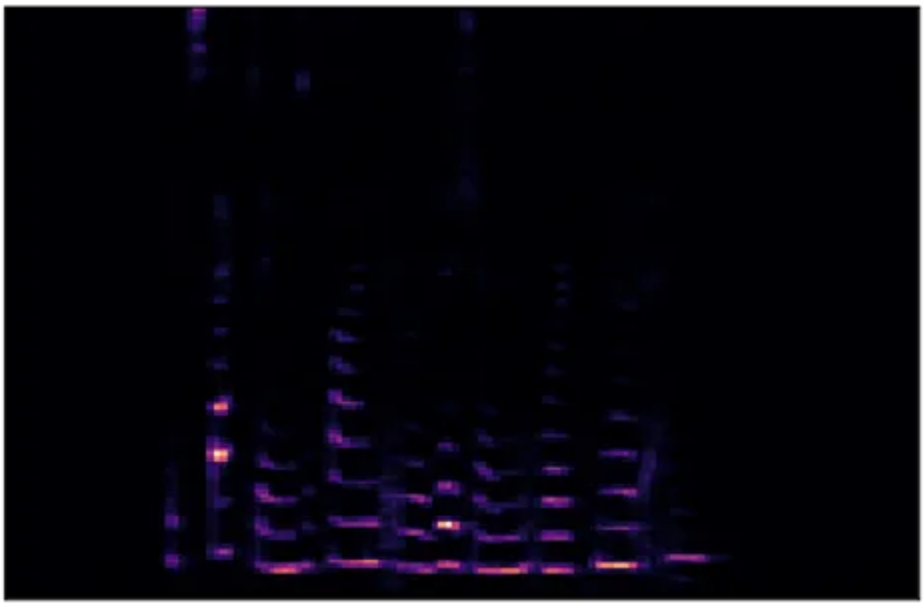
- It uses the Mel Scale instead of Frequency on the y-axis.
- It uses the Decibel Scale instead of Amplitude to indicate colors.

For deep learning models, we usually use this rather than a simple Spectrogram.

Let's modify our Spectrogram code above to use the Mel Scale in place of Frequency.

```
# use the mel-scale instead of raw frequency
sgram_mag, _ = librosa.magphase(sgram)
mel_scale_sgram = librosa.feature.melspectrogram(S=sgram_mag, sr=sample_rate)
librosa.display.specshow(mel_scale_sgram)
```

mel_scale.py hosted with ❤️ by GitHub view raw

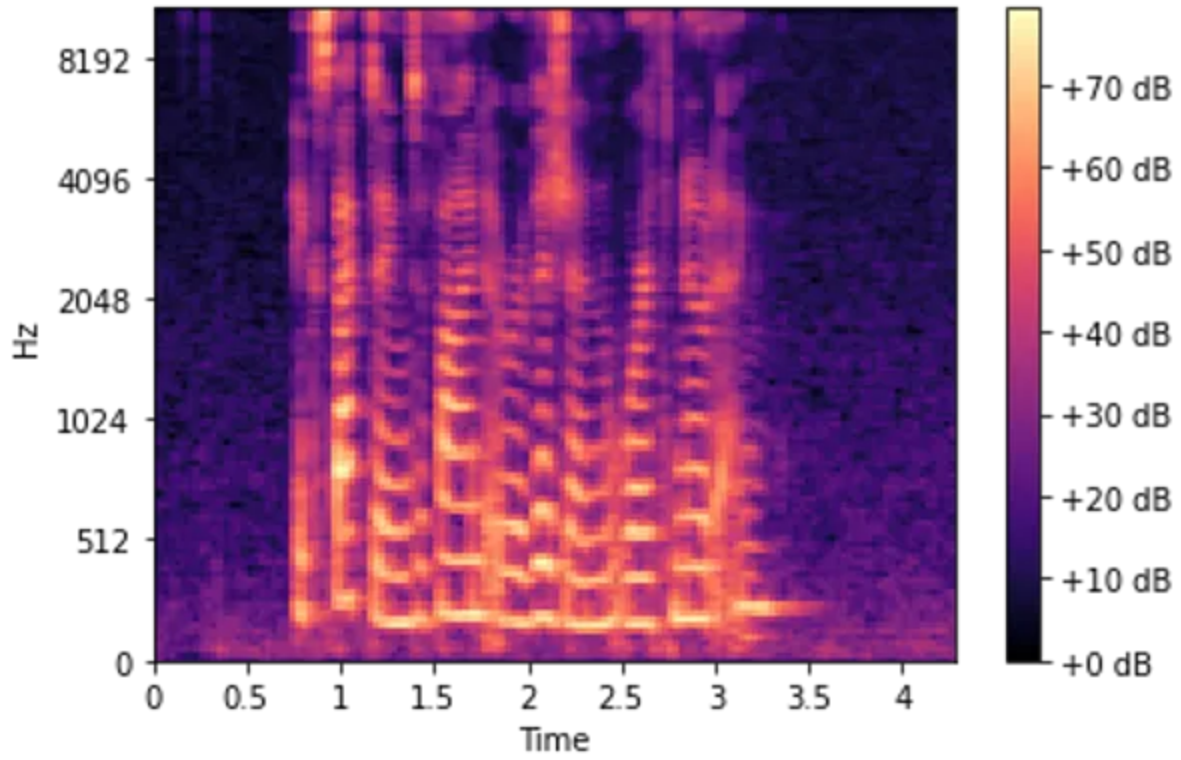


Spectrogram using Mel Scale (Image by Author)

This is better than before, but most of the spectrogram is still dark and not carrying enough useful information. So let's modify it to use the Decibel Scale instead of Amplitude.

```
# use the decibel scale to get the final Mel Spectrogram
mel_sgram = librosa.amplitude_to_db(mel_scale_sgram, ref=np.min)
librosa.display.specshow(mel_sgram, sr=sample_rate, x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
```

mel_spectrogram.py hosted with ❤️ by GitHub view raw



Mel Spectrogram (Image by Author)

Finally! This is what we were really looking for 😃.

Conclusion

We have now seen how we pre-process audio data and prepare Mel Spectrograms. But before we can input them into deep learning models, we have to optimize them to obtain the best performance.

In the next article, we will look at how we can enhance the data for our models by tuning our Mel Spectrograms, as well as augmenting our audio data to help our models generalize to a wider range of inputs.

And finally, if you liked this article, you might also enjoy my other series on Transformers, Geolocation Machine Learning, and Image Caption architectures.

Transformers Explained Visually (Part 1): Overview of Functionality

A Gentle Guide to Transformers for NLP, and why they are better than RNNs, in Plain English. How Attention helps...

towardsdatascience.com

Leveraging Geolocation Data for Machine Learning: Essential Techniques

A Gentle Guide to Feature Engineering and Visualization with Geospatial data, in Plain English

towardsdatascience.com

Image Captions with Deep Learning: State-of-the-Art Architectures

A Gentle Guide to Image Feature Encoders, Sequence Decoders, Attention, and Multi-modal Architectures, in Plain English

towardsdatascience.com

Let's keep learning!

Deep Learning | Artificial Intelligence | Data Science | Audio | Speech Recognition

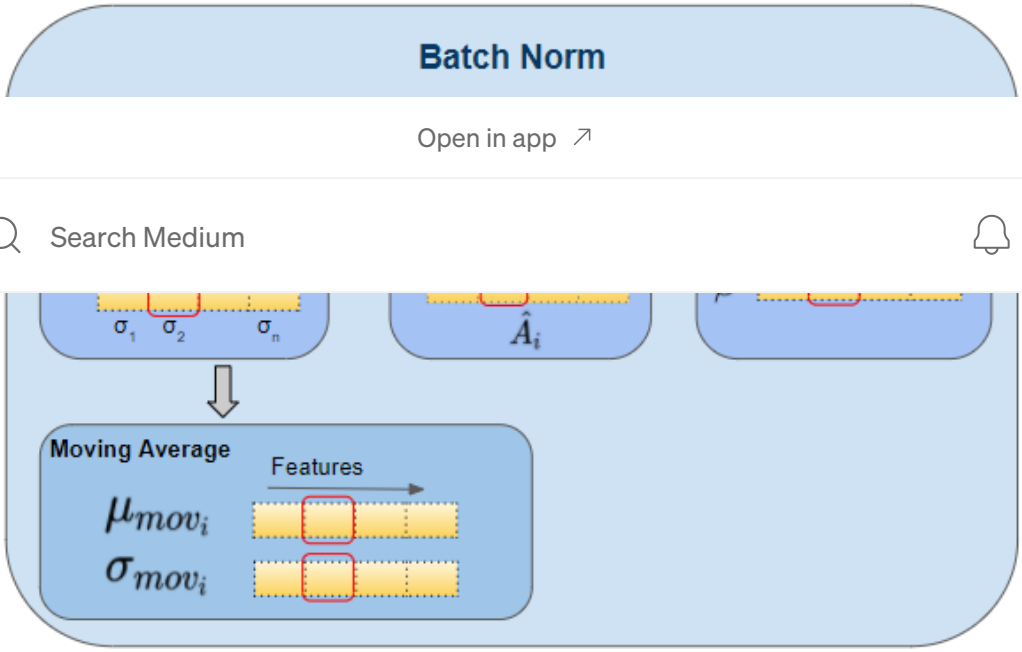
Follow

Written by Ketan Doshi

4.6K Followers · Writer for Towards Data Science

Machine Learning and Big Data

More from Ketan Doshi and Towards Data Science



Ketan Doshi in Towards Data Science

Batch Norm Explained Visually—How it works, and why neural networks need it

A Gentle Guide to an all-important Deep Learning layer, in Plain English

9 min read · May 18, 2021

Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and hopefully save you some research time and...

10 min read · Sep 17

537 11

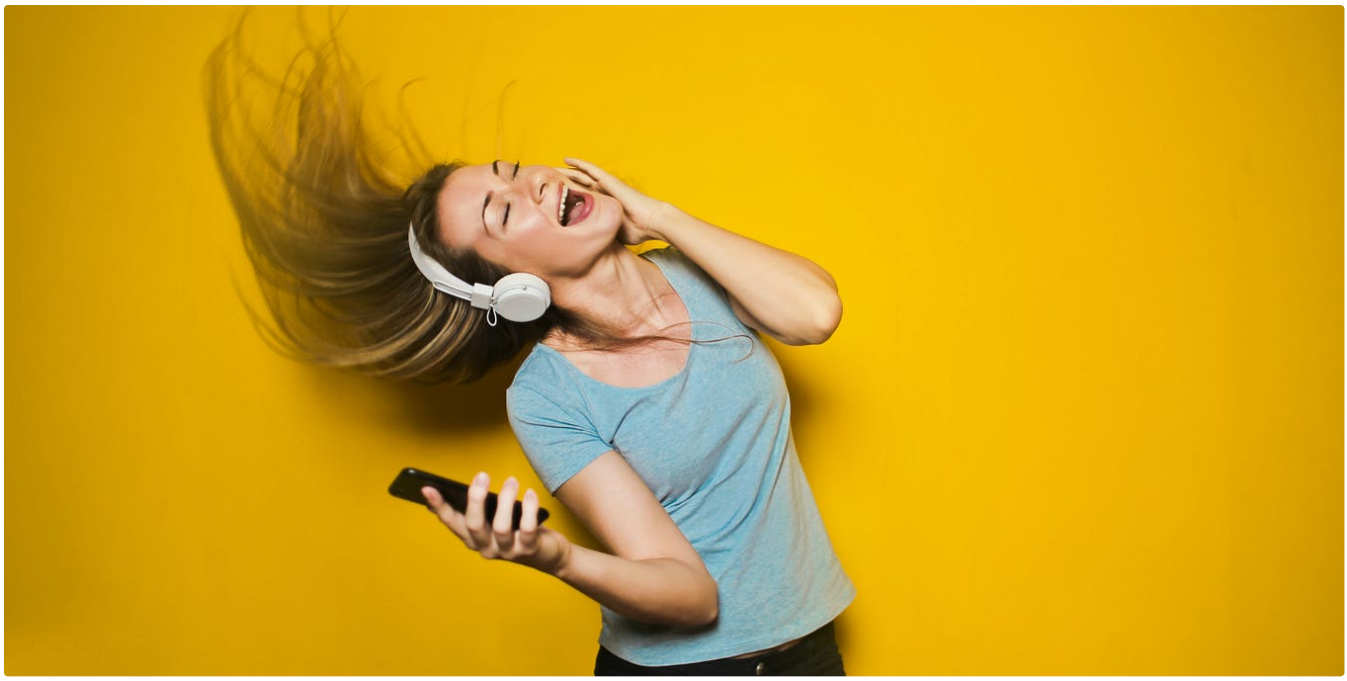
Khouloud El Alami in Towards Data Science

Don’t Start Your Data Science Journey Without These 5 Must-Do Steps From a Spotify Data Scientist

A complete guide to everything I wish I’d done before starting my Data Science journey, here’s to acing your first year with data

18 min read · 4 days ago

878 5



Ketan Doshi in Towards Data Science

Audio Deep Learning Made Simple: Sound Classification, step-by-step

An end-to-end example and architecture for Audio Deep Learning's foundational application scenario, in Plain English.

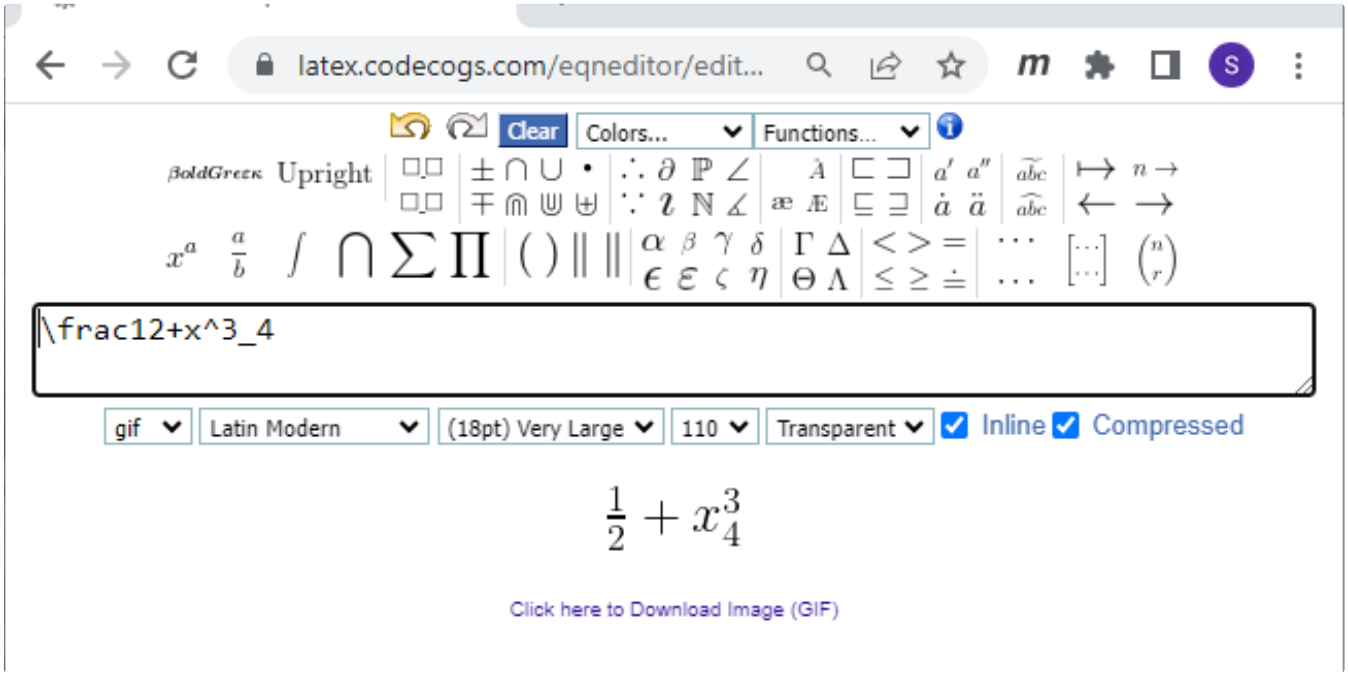
12 min read · Mar 18, 2021

Technocrat in CoderHack.com

Introduction to LibROSA

LibROSA is a Python package for audio and music analysis. It provides various functions to quickly extract key audio features and metrics...

3 min read · Sep 15



Sparisoma Viridi

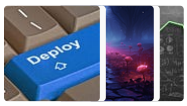
Some LaTeX equations in Markdown

LaTeX is a typesetting software used as document preparation system and functions as a plain text markup language, that leverages optimal...

5 min read · Jun 30

1

Lists



Predictive Modeling w/ Python

20 stories · 428 saves

ChatGPT prompts

24 stories · 434 saves

ChatGPT

21 stories · 169 saves

Natural Language Processing

657 stories · 259 saves

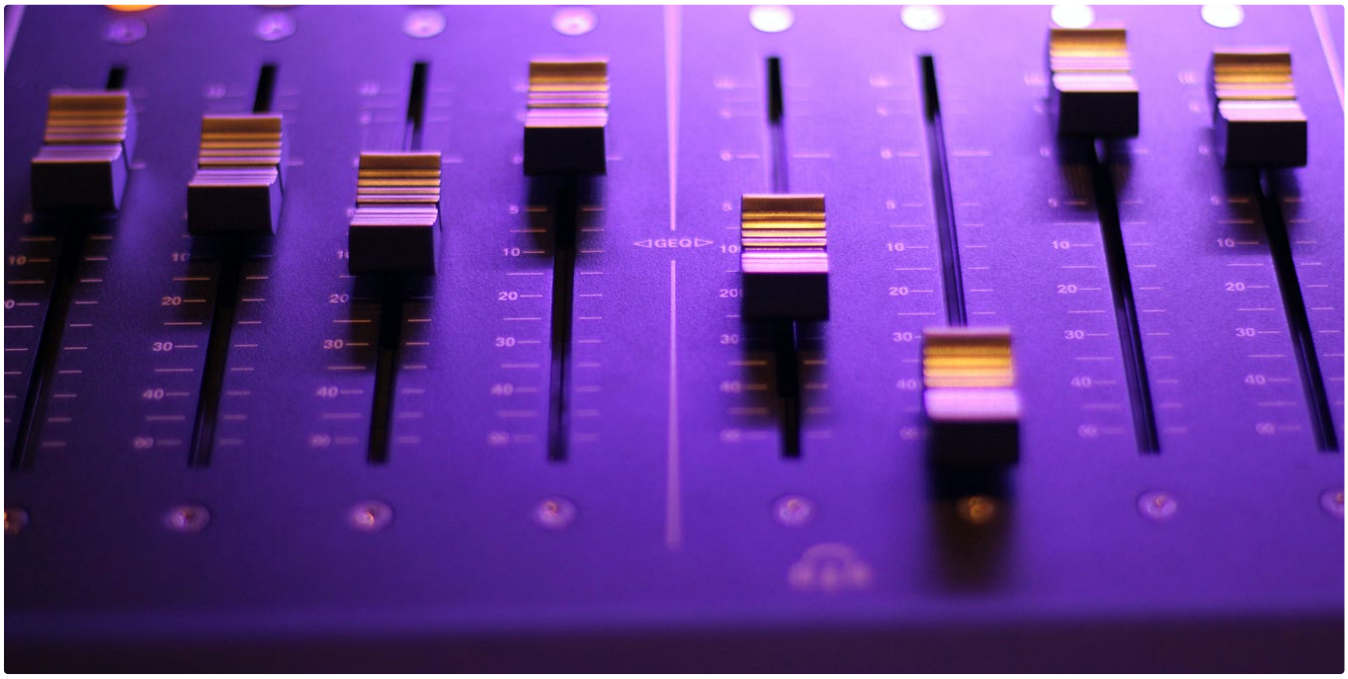
Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

· 4 min read · Feb 16, 2022

65K 947



Sharon Grundmann in ML6team

Fine-tuning Whisper for Dutch Language: The Crucial Role of Size

OpenAI has taken over the AI landscape by storm with its recent releases—yes, the title of this blog post was generated with the help of...

5 min read · Jul 19

35 1

Juanma Coria in Better Programming

Color Your Captions: Streamlining Live Transcriptions With “diart” and OpenAI’s Whisper

Combine OpenAI’s Whisper with diart for speaker-aware captions!

7 min read · Apr 18

 436  10 

Yang Zhou in TechToFreedom

9 Fabulous Python Tricks That Make Your Code More Elegant

Pythonic is a synonym for elegant

· 5 min read · Nov 14, 2022

2.7K 26

See more recommendations