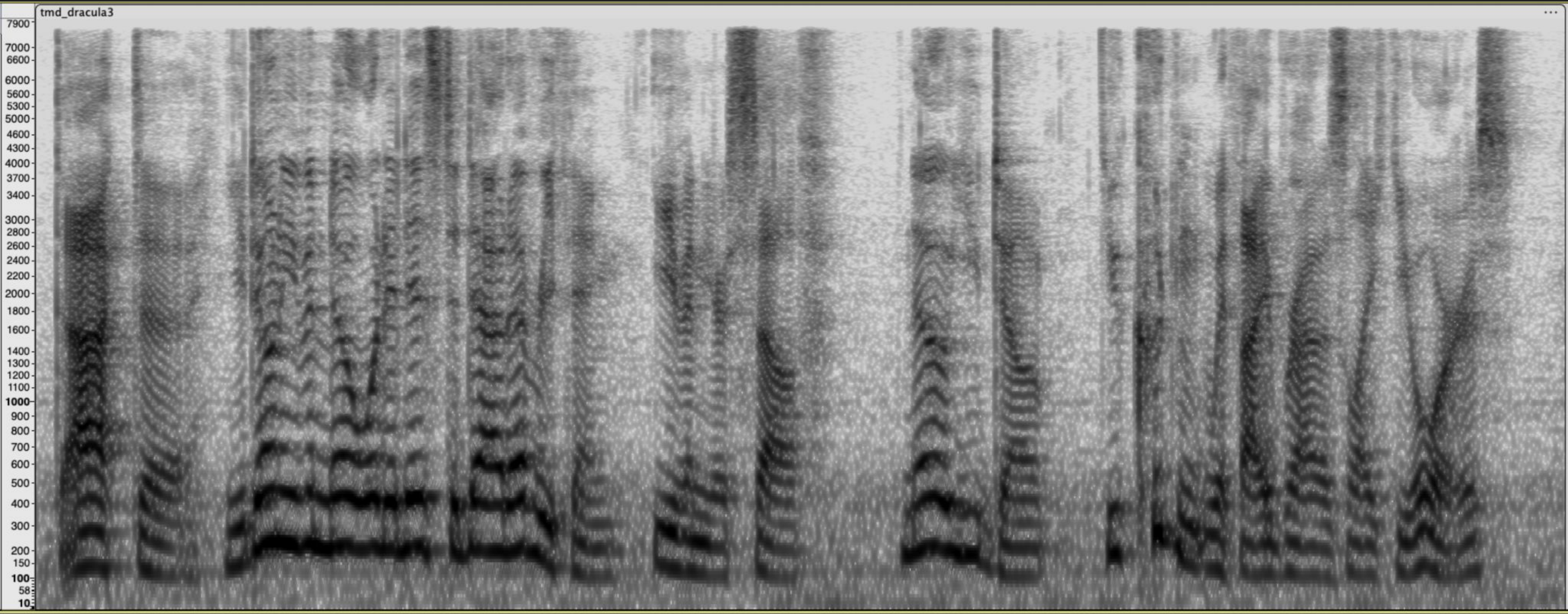
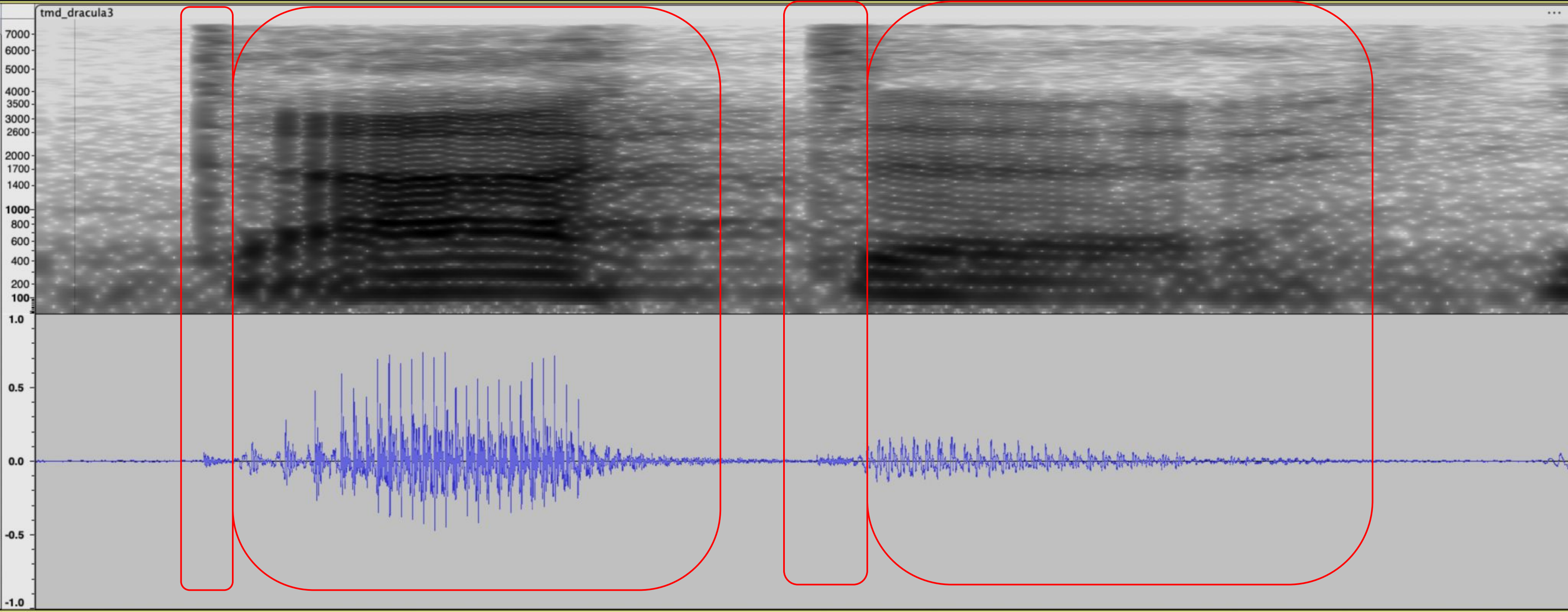
an aside on fourier transforms

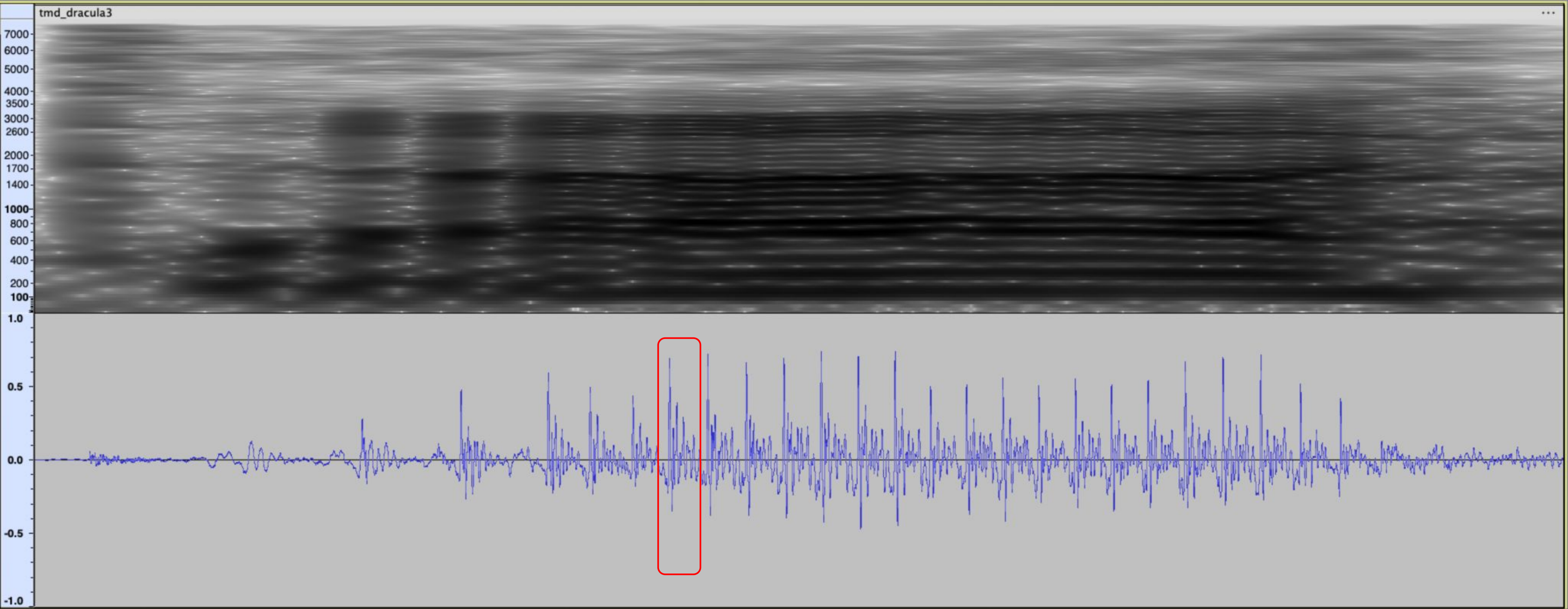
speech spectrogram example



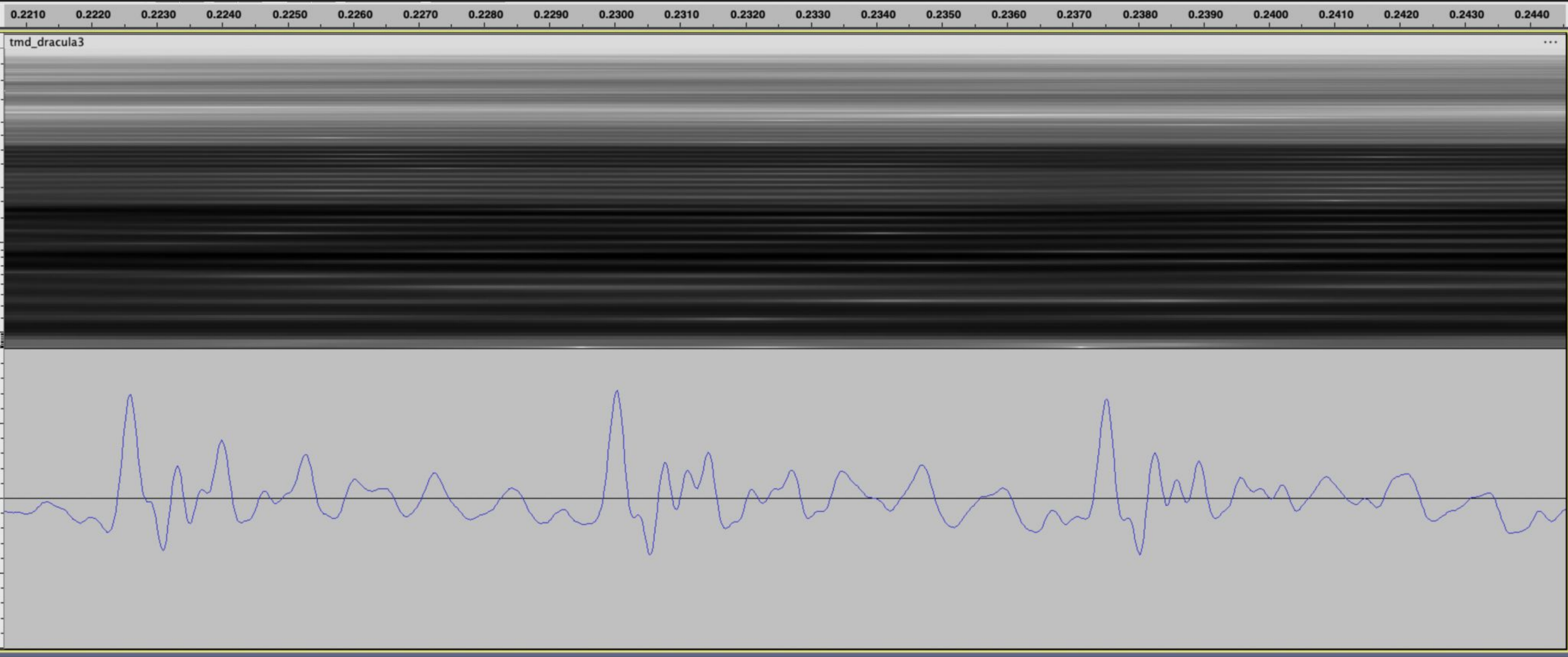
speech spectrogram example: 1s



speech spectrogram example: "dah"



speech spectrogram example: "locally periodic"



Periodic Functions

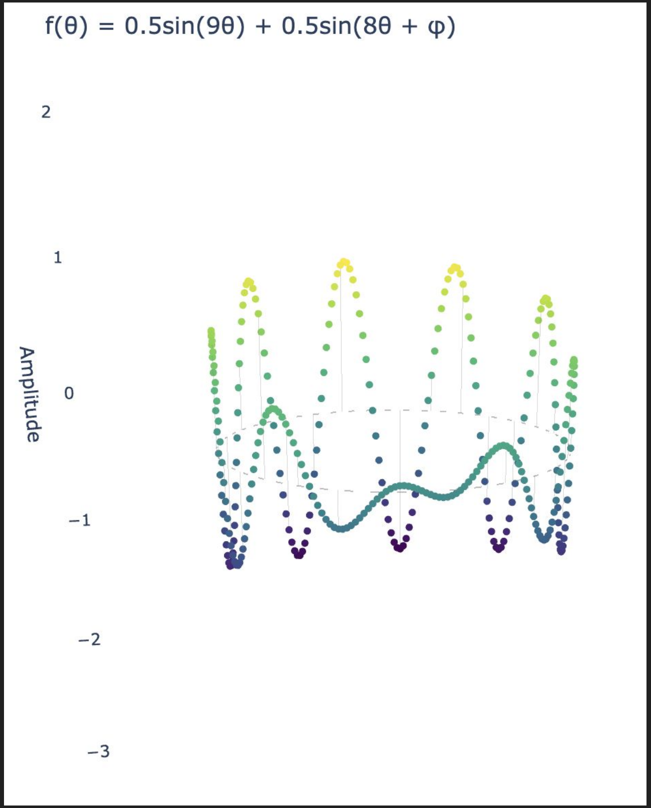
Let f: S -> C be a periodic function:
A function on the circle, giving a 'height' at each point on the circle.

The Fourier Transform is a change of basis, using specific **orthonormal basis functions**:

$$x_k(t) = e^{-i*2\pi*k*t}$$

Then $\mathbf{f_k = \int f(t)*x_k(t)}$ gives the nth component.

(Note: This is the continuous equivalent of a dot product of a vector with each new basis vector.)



Discrete Periodic Functions

I find it easier to think about the discrete case!
(but I'm a combinatorialist at heart, ymmv.)

In this case, we have functions Z_N->R.
(If N is big, it starts looking like the continuous case.)
These *functions* can be written as *arrays*!

One basis for these functions is the elementary basis: [0, 0, …, 0, 1, 0, …, 0].
This is how we represent the time domain:
Sample by sample.

The Fourier Transform is a change of basis, using specific **orthonormal basis functions** specifically chosen to measure periodicity.

Discrete Fourier transform

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \frac{k}{N} n} \qquad \textbf{(Eq.1)}$$

The raw DFT is complex-valued, so each frequency component has two parts: **magnitude** and **phase**.

When making a visual spectrogram,
we keep only the magnitude, and discard phase.

You can use the Griffin-Lim algorithm to try to reconstruct phase from a spectrogram.
…Or WaveNet!

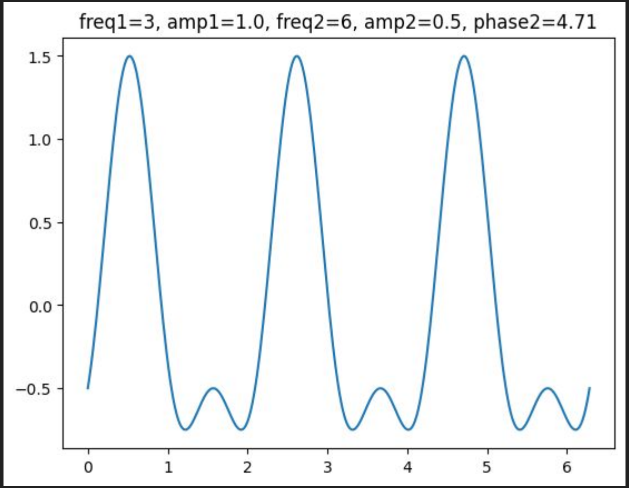
Sinusoidal functions are a basis for periodic functions.

Similar to a Taylor series…

You can express any* periodic function as a **linear combination of sinusoids** (with different frequencies, amplitudes, and phases).

The sinusoidal basis functions are 'spread out' in the time domain, but 'local' in the frequency domain.

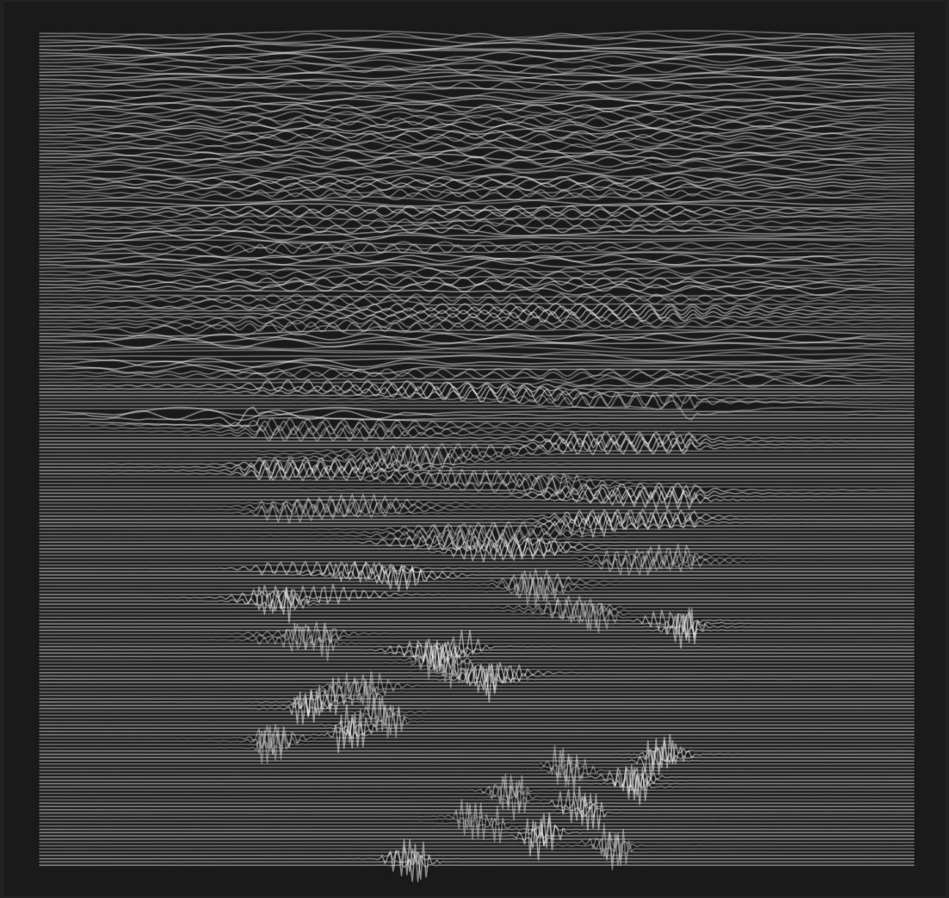
Likewise, the sample basis is 'local' in the time domain, but 'spread out' in the frequency domain.



…but the fft isn't the only way to represent audio.

Another major approach is wavelets,
which use variable length audio windows to try
to localize shorter higher-frequency "chirps".

You can also use a big convolution and try to learn good basis functions directly, like the set pictured at right. These learned functions are very similar to wavelets!



wrapping up

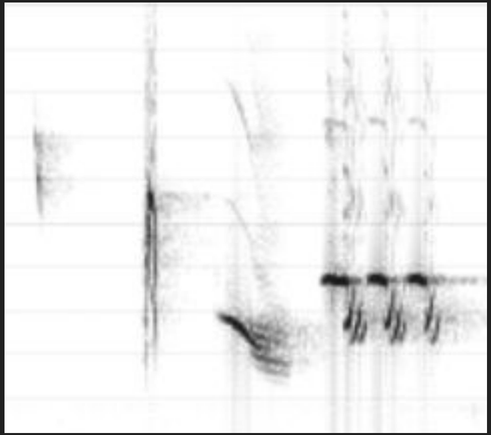
colab notebook 2…

Basic audio data processing

Hopefully, we learned some basic operations:

- Extracting spectrograms from a long recording,
- Reading a CSV of time-window annotations and matching them with different time-windows.

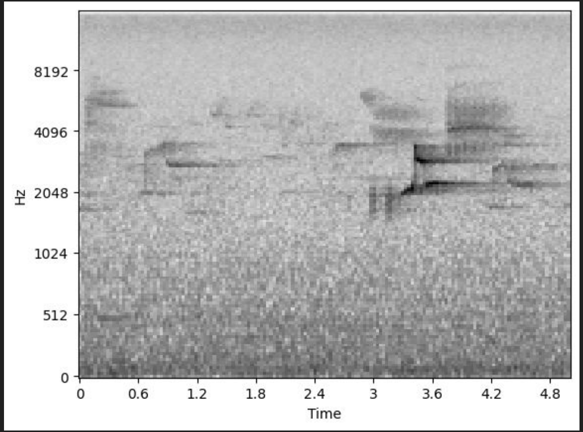
We also tried learning directly from aggregated spectrogram features…

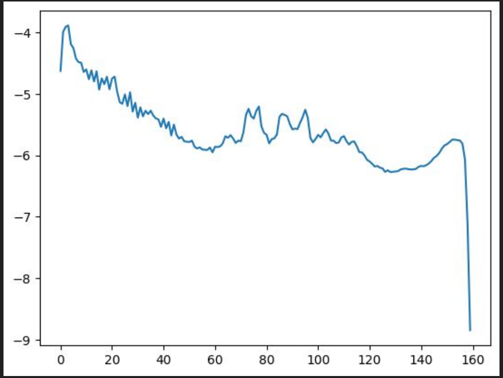


Raw Spectral Features Aren't Enough

We know that species vocalize in some range of frequencies, buuuut:

- Amplitude will change across recordings (distance!)
- Other species/noise might occur in those frequencies (leading to false positives)



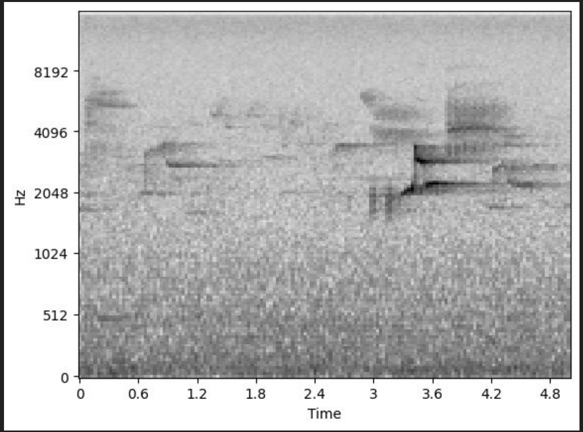


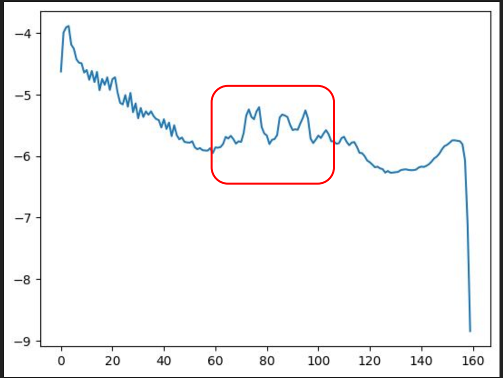
Raw Spectral Features Aren't Enough

When we take (say) the mean of the spectral energies over time, we lose all of the 'shape' information in the spectrogram.

A lot of information is lost!

A convolutional network is great for learning shapes, however…





perch hoplite overview

Perch Hoplite

Inference (and tool) library to allow efficient application of models to large bioacoustic datasets.

Available on github! (QR-Code goes there.)

Why "Hoplite"?
Because there's a **vector database**,
which 'hops' around a graph.
Also, it's lightweight.
(Also also, tmd@ likes painting hoplites.)

Four Sub-Libraries…

- **taxonomy**
  tools for handling different bird/label taxonomies.

- **zoo**
  handling different kinds of audio model.

- **db**
  vector and metadata database for big datasets.

- **agile**
  tools for classifier development.

taxonomy

There are three global scale bird taxonomies, with sliiiightly different naming and choices.

Two of these taxonomies update yearly.

There are also many different common names and regional taxonomies (eg, North American four-letter bird codes).

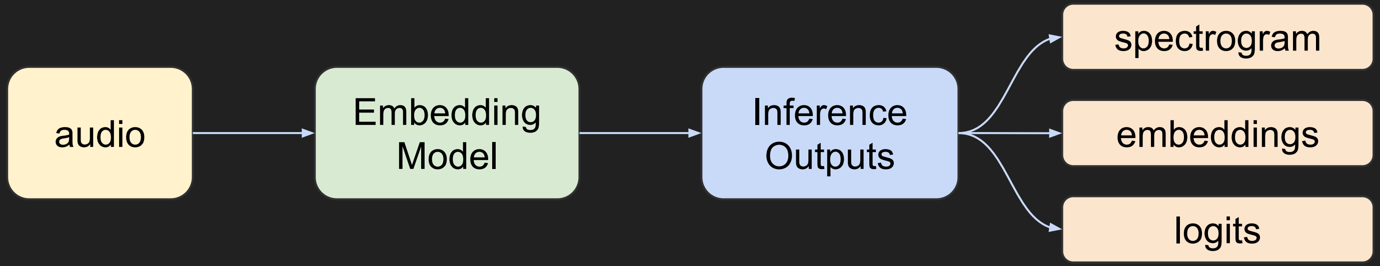
Each dataset uses one of these taxonomies… or does ad hoc naming.

The taxonomy library handles conversion between these different sets of labels.

Three pieces:

- **Namespace** - a named set of strings.
- **Mapping** - a conversion from one namespace to another.
- **Class List** - an ordered subset of strings in a specific namespace. (For example, classifier outputs!)

ZOO



The Zoo library is a model 'warehouse.'

`EmbeddingModel` is the core interface abstraction. We can make an implementation for any model which consumes audio and outputs some combination of embeddings and class logits.

The main method is a **`model.embed`**, which takes in arbitrary-length audio and outputs an **`InferenceOutputs`** object, with embeddings, logits, and spectrogram.

- Each `EmbeddingModel` implementation is responsible for doing any audio frontend conversion needed. This simplifies things for downstream use, since every model needs different frontend stuff.
- Each implementation is also responsible for slicing/framing/normalizing audio into windows, as needed. However, some tools are provided.
- Returning the frontend outputs is helpful for debugging!

`zoo` - one gory detail.

Generally, machine learning models run **much faster** if they are evaluated on a batch of audio. But some models only work on single instances…

The EmbeddingModel interface has both **`embed`** and **`batch_embed`**:
You only need to implement one, and then use a converter for the other (embed_from_batch_embed or batch_embed_from_embed).

Given a batch of audio of the same length [N, s], for Perch we:

- Pad the audio to a multiple of 5s: [N, k*5*32_000]
- Reshape the audio into one big batch: [N*k, 5*32_000]
- Feed it to the model.
- Get spectrograms [N*k, 500, 128], embeddings [N*k, 1536] and logits [N*k, 15k].
- Reshape to include window dimension: [N, k, 500, 128] for spectrograms.

db: The beating heart of everything…

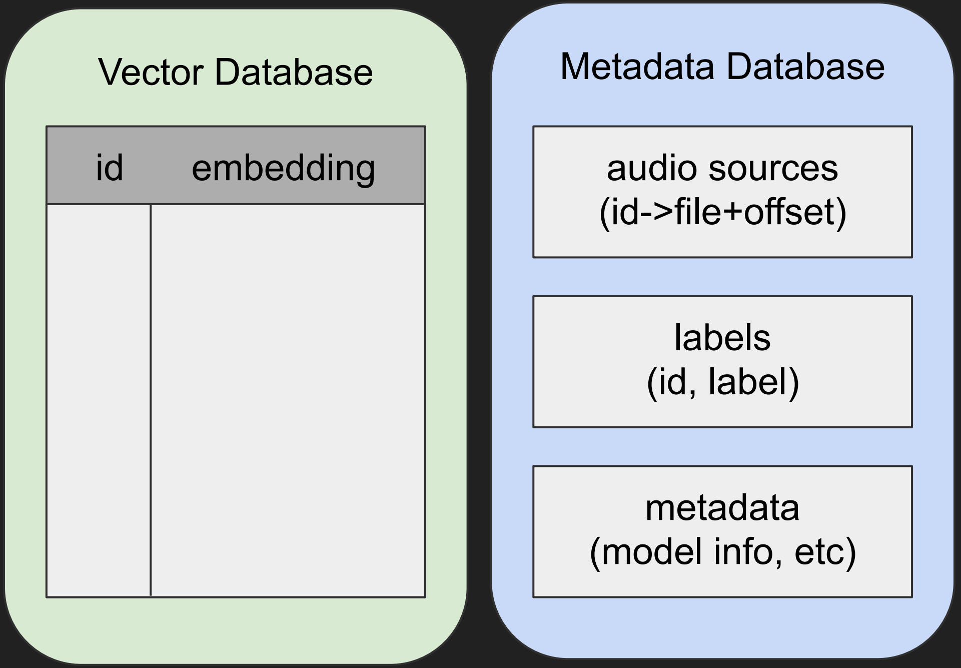
A database for managing a combination of embedding vectors and metadata connecting the embedding vectors back to actual audio.

Two essential parts: **Vector database** and **metadata database**.

Like the zoo, we have an interface for the db, allowing multiple implementations. Right now, there are two implementations:

- Dicts+numpy
- SQLite+Usearch

We'll do a whole discussion on vector databases soon…



db: One gory feature…

For audio locations, we have a 'base path' stored in the metadata, and then each embedding source is stored relative to the base path.

This lets us move all of the audio to somewhere new (eg, desktop->cloud), and only update one thing (the base path) in the database metadata.

db: Why SQLite?

SQLite is surprisingly awesome.

Pros:

- Can handle immense amounts of data.
- Extremely fast.
- Battle hardened: Billions of users from web servers to android apps.
- No connection/auth overhead.
- Single writer / multiple reader is great for our use case.

Cons:

- Cannot access remote data.

agile

Everything "user-facing"...

- **`embed`** - Embed lots of audio and place the embeddings in the database.

- **classify** - Train linear classifiers on embeddings. Make train/test splits from labeled (and unlabeled!) data in the database.

- **embedding_display** - Make pretty pictures and label buttons in Colab.

Also two Colab notebooks which bring everything together:

- `1_embed_audio_v2.ipynb`
- `2_agile_modeling_v2.ipynb`