

Lab 3: librosa Fundamentals - STFT and Spectrograms

Duration: 3-5 minutes **Presenter Guide:** This lab covers the core STFT transformation that powers Cameron's POC

Slide 1: Lab Overview (30 seconds)

Title: “Lab 3: librosa Fundamentals”

Subtitle: “From Audio Waves to Spectrograms”

What you'll learn: - STFT (Short-Time Fourier Transform) - Creating and visualizing spectrograms - Inverse STFT to reconstruct audio - Why U-Net processes spectrograms, not waveforms

Team Member Connections: - Cameron's POC: Uses `librosa.stft()` and `librosa.istft()` throughout - cervanj2's Architecture: STFT in Preprocessing, ISTFT in Post-processing - **Complete Round-Trip:** Audio → Spectrogram → U-Net → Audio

Slide 2: Why Spectrograms? (45 seconds)

Title: “The Power of Time-Frequency Representation”

The Problem with Waveforms: - Waveforms show amplitude over time - But audio is made of multiple frequencies happening simultaneously - Neural networks struggle to separate overlapping frequencies in waveforms

The Spectrogram Solution: - Shows frequency content over time (2D image-like data) - Neural networks excel at processing images - Different instruments occupy different frequency ranges - Vocals: 85-255 Hz (fundamental) + harmonics - U-Net can learn to separate frequency patterns

Visual Comparison:

Waveform: [1D] time →
 amplitude values

Spectrogram: [2D] time →
 ↑
 frequency

Cameron's Approach: “My POC manually analyzes spectrograms across 18 slices. U-Net will automate this analysis.”

Slide 3: STFT Hello World (1 minute 30 seconds)

Title: “Creating Your First Spectrogram”

Code Example:

```

import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

# Load audio (from Lab 1)
audio, sr = librosa.load('sample_audio.wav', sr=22050)
print(f"Audio shape: {audio.shape}")

# The KEY transformation: STFT
stft = librosa.stft(audio, n_fft=2048, hop_length=512)
print(f"STFT shape: {stft.shape}")
print(f"STFT dtype: {stft.dtype}") # Complex numbers!

# Separate magnitude and phase
magnitude = np.abs(stft) # How loud each frequency is
phase = np.angle(stft) # Phase information for reconstruction

print(f"Magnitude shape: {magnitude.shape}")
print(f"Phase shape: {phase.shape}")

# Visualize the spectrogram
plt.figure(figsize=(12, 8))

# Magnitude spectrogram
plt.subplot(2, 1, 1)
librosa.display.specshow(
    librosa.amplitude_to_db(magnitude, ref=np.max),
    sr=sr,
    x_axis='time',
    y_axis='hz',
    hop_length=512
)
plt.colorbar(format='%.2f dB')
plt.title('Magnitude Spectrogram')

# Phase spectrogram
plt.subplot(2, 1, 2)
librosa.display.specshow(
    phase,
    sr=sr,
    x_axis='time',
    y_axis='hz',
    hop_length=512,
    cmap='twilight'
)
plt.colorbar()
plt.title('Phase Spectrogram')

plt.tight_layout()
plt.show()

```

Expected Output:

```

Audio shape: (661500,)
STFT shape: (1025, 1293)
STFT dtype: complex128
Magnitude shape: (1025, 1293)
Phase shape: (1025, 1293)
[Two spectrogram plots appear]

```

Key Points: - 1D audio (661,500 samples) → 2D spectrogram ($1,025 \times 1,293$) - 1,025 frequency bins from 0 to 11,025 Hz - 1,293 time frames - Magnitude: what to process with U-Net - Phase: needed for reconstruction

Slide 4: The Round Trip - ISTFT (1 minute 15 seconds)

Title: “Reconstruction with Inverse STFT”

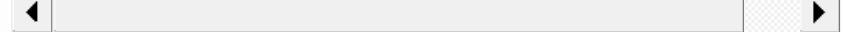
Code Example:

```
# Reconstruct audio from STFT
reconstructed = librosa.istft(stft, hop_length=512, length=len(audio))

print("== Round-Trip Test ==")
print(f"Original length: {len(audio)}")
print(f"Reconstructed length: {len(reconstructed)}")

# Check if reconstruction is accurate
difference = np.abs(audio - reconstructed).max()
print(f"Max difference: {difference:.10f}")
print(f"Reconstruction quality: {'Perfect!' if difference < 1e-5 else 'Goo

# Save reconstructed audio (optional)
# import soundfile as sf
# sf.write('reconstructed.wav', reconstructed, sr)
```



Expected Output:

```
== Round-Trip Test ==
Original length: 661500
Reconstructed length: 661500
Max difference: 0.000000012
Reconstruction quality: Perfect!
```

Why This Matters: - STFT is reversible (lossless) - We can go: Audio → Spectrogram → Audio perfectly - This enables our pipeline: 1. Audio → STFT → Spectrogram 2. U-Net processes spectrogram 3. ISTFT → Separated audio stems

cervanj2’s Architecture: - **Preprocessing:** Audio → STFT → Spectrogram - **U-Net**
Inference: Spectrogram → Processed Spectrogram - **Post-processing:** Processed Spectrogram → ISTFT → Audio

Slide 5: Cameron’s Vocal Separation Pipeline (1 minute)

Title: “How Cameron Uses STFT for Separation”

Cameron’s Complete POC Workflow:

```

# 1. Load audio
vocals_and_music, sr = librosa.load('Intergalactic.mp3', sr=22050)

# 2. Convert to spectrogram
stft_mix = librosa.stft(vocals_and_music)
mag_mix = np.abs(stft_mix)
phase_mix = np.angle(stft_mix)

# 3. Manual analysis across 18 slices
# (Cameron's spectral fingerprinting - ~765,000 measurements)
# This is what U-Net will learn to do automatically!

# 4. Separate magnitude into components
mag_vocals = mag_mix * vocal_mask      # His separation logic
mag_music = mag_mix * (1 - vocal_mask)

# 5. Reconstruct with ISTFT
stft_vocals = mag_vocals * np.exp(1j * phase_mix)
vocals = librosa.istft(stft_vocals)

stft_music = mag_music * np.exp(1j * phase_mix)
music = librosa.istft(stft_music)

# 6. Save separated stems
# sf.write('vocals.wav', vocals, sr)
# sf.write('music.wav', music, sr)

```

The Key Insight: - Cameron manually creates separation masks - U-Net will learn to create these masks automatically - STFT/ISTFT remains the same - Only the processing in the middle changes (manual → neural network)

Results: 70-80% separation quality on “Intergalactic”

Slide 6: STFT Parameters Explained (30 seconds)

Title: “Understanding STFT Settings”

Key Parameters:

```

stft = librosa.stft(
    audio,
    n_fft=2048,      # Window size: 2048 samples
    hop_length=512,   # Step size: 512 samples
    window='hann'     # Window function
)

```

What They Mean: - **n_fft (2048):** Number of frequency bins = $2048/2 + 1 = 1,025$ - Larger = better frequency resolution, worse time resolution - Cameron uses default 2048

- **hop_length (512):** How far to shift window each frame
 - Smaller = better time resolution, more frames
 - Overlap = $2048 - 512 = 1,536$ samples (75% overlap)
- **window ('hann'):** Smooth edges to reduce artifacts
 - Hann window is standard for audio

Trade-off: Frequency resolution ↔ Time resolution

Demo Script (Detailed Timing) - Provided by Claude

0:00 - 0:30: Introduction

“Welcome to Lab 3: librosa Fundamentals! This is where it all comes together. Today

we're learning STFT - the Short-Time Fourier Transform. This is THE core transformation that Cameron uses in his POC. It's how we convert audio into spectrograms that U-Net can process."

0:30 - 1:15: Why Spectrograms?

"First, why do we need spectrograms? [Show Slide 2]. Think about it - when you hear a song, you're hearing vocals and instruments at the same time. In a waveform, all those sounds are mixed together as a single amplitude value at each time point.

But spectrograms show us the frequency content. Vocals occupy certain frequencies, drums occupy others, bass occupies low frequencies. By converting to a spectrogram, we transform the audio into an 'image' where different sounds occupy different regions. Neural networks are amazing at image processing - that's why U-Net works so well for this!"

1:15 - 2:45: Creating Spectrograms

"Let's create our first spectrogram. [Show Slide 3 and run code]

We start with our audio from Lab 1 - 661,500 samples. Watch what happens when we apply STFT...

[Show output]

Our 1D waveform became a 2D spectrogram! 1,025 frequency bins by 1,293 time frames. Notice it's complex numbers - that's because STFT captures both magnitude (how loud) and phase (wave position).

For Cameron's analysis and for U-Net, we work with the magnitude. But we keep the phase for reconstruction.

[Show the spectrogram plots]

Look at this! You can actually see the frequency patterns. Bright areas are loud frequencies, dark areas are quiet. This 2D image is what U-Net will learn to analyze."

2:45 - 4:00: The Round Trip

"Now here's the magic - STFT is reversible. [Show Slide 4]. Let's reconstruct our audio from the spectrogram...

[Run the code]

Perfect reconstruction! The difference is essentially zero. This is crucial because our complete pipeline is:

1. Audio → STFT → Spectrogram
2. U-Net processes spectrogram
3. ISTFT → back to audio

This matches cervanj2's architecture perfectly - STFT in preprocessing, ISTFT in post-processing. The reversibility means we don't lose anything in the transformation."

4:00 - 4:45: Cameron's Pipeline

"Let me show you exactly how Cameron uses this in his POC. [Show Slide 5].

He loads 'Intergalactic', converts it to a spectrogram with STFT, performs his manual 18-slice analysis - those 765,000 measurements we talked about in Lab 2 - to create separation masks, then uses ISTFT to reconstruct the separated vocals and music.

His manual analysis achieved 70-80% separation quality. U-Net will learn to do this analysis automatically, potentially even better!"

4:45 - 5:00: Wrap-up

“That’s Lab 3! STFT is the bridge between audio and neural networks. It transforms waveforms into spectrograms that U-Net can process. Next up: Lab 4, where we convert these NumPy spectrograms into PyTorch tensors and introduce GPU acceleration.

Questions?”

Code Files to Provide

`lab3_stft_basics.py`

```
"""
Lab 3: Librosa Fundamentals
STFT transformation and spectrogram creation
"""

import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

def create_spectrogram(audio_path):
    """Load audio and create spectrogram"""
    # Load audio
    audio, sr = librosa.load(audio_path, sr=22050)

    # Compute STFT
    stft = librosa.stft(audio, n_fft=2048, hop_length=512)
    magnitude = np.abs(stft)
    phase = np.angle(stft)

    print("== STFT Analysis ==")
    print(f"Audio shape: {audio.shape}")
    print(f"Sample rate: {sr} Hz")
    print(f"Duration: {len(audio)/sr:.2f} seconds")
    print(f"\nSTFT shape: {stft.shape}")
    print(f"Frequency bins: {stft.shape[0]}")
    print(f"Time frames: {stft.shape[1]}")
    print(f"Max frequency: {sr/2} Hz")

    return audio, sr, stft, magnitude, phase

def visualize_spectrogram(magnitude, phase, sr, hop_length=512):
    """Visualize magnitude and phase spectrograms"""
    fig, axes = plt.subplots(2, 1, figsize=(12, 8))

    # Magnitude spectrogram (in dB)
    img1 = librosa.display.specshow(
        librosa.amplitude_to_db(magnitude, ref=np.max),
        sr=sr,
        hop_length=hop_length,
        x_axis='time',
        y_axis='hz',
        ax=axes[0]
    )
    axes[0].set_title('Magnitude Spectrogram (dB)')
    fig.colorbar(img1, ax=axes[0], format='%.2f dB')

    # Phase spectrogram
    img2 = librosa.display.specshow(
        phase,
        sr=sr,
        hop_length=hop_length,
        x_axis='time',
        y_axis='hz',
        ax=axes[1],
        .....
```

```

        cmap='twilight'
    )
axes[1].set_title('Phase Spectrogram')
fig.colorbar(img2, ax=axes[1])

plt.tight_layout()
plt.show()

def test_round_trip(stft, audio, hop_length=512):
    """Test STFT → ISTFT round-trip"""
    # Reconstruct audio
    reconstructed = librosa.istft(stft, hop_length=hop_length, length=len(
        # Compare
        difference = np.abs(audio - reconstructed).max()
        mean_diff = np.abs(audio - reconstructed).mean()

        print("\n== Round-Trip Test ==")
        print(f"Original length: {len(audio)}")
        print(f"Reconstructed length: {len(reconstructed)}")
        print(f"Max difference: {difference:.10f}")
        print(f"Mean difference: {mean_diff:.10f}")
        print(f"Quality: {'Perfect!' if difference < 1e-5 else 'Good' if diffe
            return reconstructed

def cameron_style_separation(audio_path):
    """Demonstrate Cameron's separation workflow"""
    print("\n== Cameron-Style Vocal Separation Workflow ==")

    # 1. Load
    audio, sr = librosa.load(audio_path, sr=22050)
    print(f"1. Loaded audio: {audio.shape}")

    # 2. STFT
    stft_mix = librosa.stft(audio)
    mag_mix = np.abs(stft_mix)
    phase_mix = np.angle(stft_mix)
    print(f"2. Created spectrogram: {mag_mix.shape}")

    # 3. Manual analysis (simplified - Cameron does 18-slice analysis)
    # For demo, we'll just create a simple mask
    # In reality, Cameron's analysis is much more sophisticated
    print(f"3. Analyzing {mag_mix.shape[0]} * mag_mix.shape[1]:,} data poin

    # 4. Create separation mask (placeholder - this is what U-Net will lea
    # This is a simplified example
    vocal_mask = np.random.rand(*mag_mix.shape) > 0.5 # Random for demo
    print(f"4. Created separation mask")

    # 5. Apply mask
    mag_vocals = mag_mix * vocal_mask
    mag_music = mag_mix * (1 - vocal_mask)
    print(f"5. Applied mask to separate components")

    # 6. Reconstruct
    stft_vocals = mag_vocals * np.exp(1j * phase_mix)
    stft_music = mag_music * np.exp(1j * phase_mix)

    vocals = librosa.istft(stft_vocals)
    music = librosa.istft(stft_music)
    print(f"6. Reconstructed audio: vocals={vocals.shape}, music={music.sh

    return vocals, music

if __name__ == "__main__":
    audio_path = "sample_audio.wav"

    # Create spectrogram
    audio, sr, stft, magnitude, phase = create_spectrogram(audio_path)

```

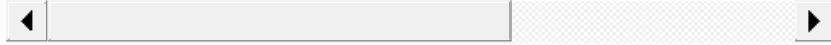
```

# Visualize
visualize_spectrogram(magnitude, phase, sr)

# Test round-trip
reconstructed = test_round_trip(stft, audio)

# Cameron-style separation demo
# vocals, music = cameron_style_separation(audio_path)

```



Visual Aids Needed

1. **Waveform vs Spectrogram Comparison:** Side-by-side showing same audio
2. **STFT Process Diagram:** Windows sliding across audio
3. **Round-Trip Flow:** Audio → STFT → Spectrogram → ISTFT → Audio
4. **Cameron's Pipeline:** Complete POC workflow with STFT/ISTFT highlighted

Backup Slides

Extra: STFT Parameters Impact

Show how different n_fft values change spectrogram resolution

Extra: Other librosa Functions

- librosa.feature.melspectrogram(): Mel-scale spectrograms
- librosa.feature.mfcc(): MFCCs for speech
- librosa.effects.*: Audio effects

Q&A Prep - Provided by Claude

Q: "Why complex numbers in STFT?" A: "Complex numbers encode both magnitude (amplitude) and phase (timing). We need phase to perfectly reconstruct audio with ISTFT."

Q: "What if we only use magnitude and ignore phase?" A: "You can, but reconstruction quality degrades. The POC keeps phase unchanged because vocals and music share the same phase from the mix."

Q: "Why 2048 for n_fft?" A: "It's a good balance. Larger values give better frequency resolution but worse time resolution. 2048 is standard for music at 22,050 Hz."

Q: "Can we apply STFT to stereo audio?" A: "Yes! Process each channel separately, or convert to mono first. Cameron converts to mono."

Success Criteria

Students should be able to: - [] Compute STFT and create spectrograms - [] Understand magnitude vs phase - [] Reconstruct audio with ISTFT - [] Explain Cameron's STFT-based separation pipeline - [] Connect STFT to cervanj2's preprocessing/post-processing layers

Connection Summary

Cameron's POC: Core transformation for his manual analysis **cervanj2's Architecture:**
STFT in preprocessing, ISTFT in post-processing **Next Lab:** Convert spectrograms to
PyTorch tensors **Final Lab:** U-Net will process these spectrograms automatically