

Step 1: Install dependencies

```
!pip install librosa soundfile --quiet
```

Step 2: Import libraries

```
import os
import io
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
from google.colab import files
from IPython.display import Audio, display
```

Step 3: Upload up to 11 .wav files manually

```
# After running, Colab will open a file picker. Select the files (max 11).
# The uploaded dict maps filename -> bytes. We'll save them into /content/uploads/.

data_dir = None
MAX_FILES = 11
upload_folder = '/content/uploads'
os.makedirs(upload_folder, exist_ok=True)

if data_dir is None:
    print(f"Please upload up to {MAX_FILES} audio files (wav). Use Ctrl/Cmd to select multiple files.")
    uploaded = files.upload()
    filenames = []
    for fname, filebytes in uploaded.items():
        if len(filenames) >= MAX_FILES:
            print(f"Ignoring extra file: {fname} (already reached {MAX_FILES}).")
            break
        # save to upload_folder
        target_path = os.path.join(upload_folder, fname)
        with open(target_path, 'wb') as f:
            f.write(filebytes)
        filenames.append(target_path)
    if len(filenames) == 0:
        raise SystemExit("No files uploaded. Please upload at least one .wav file.")
else:
    # Read wav files from data_dir
    all_files = []
```

```

for root, _, files_list in os.walk(data_dir):
    for f in files_list:
        if f.lower().endswith('.wav'):
            all_files.append(os.path.join(root, f))
all_files = sorted(all_files)
filenames = all_files[:MAX_FILES]
if len(filenames) == 0:
    raise SystemExit(f"No .wav files found in {data_dir}")

print(f"Files to process ({len(filenames)}):")
for p in filenames:
    print(" -", p)

```

Please upload up to 11 audio files (wav). Use Ctrl/Cmd to select multiple files.

<IPython.core.display.HTML object>

```

Saving [cel][cla]0001_1.wav to [cel][cla]0001_1 (4).wav
Saving [cla][cla]0150_1.wav to [cla][cla]0150_1 (3).wav
Saving [flu][cla]0346_1.wav to [flu][cla]0346_1 (3).wav
Saving [gac][cla]0518_1.wav to [gac][cla]0518_1 (3).wav
Saving [gel][cla]0901_1.wav to [gel][cla]0901_1 (3).wav
Saving [org][jaz_blu]1039_1.wav to [org][jaz_blu]1039_1 (3).wav
Saving [pia][cla]1283_1.wav to [pia][cla]1283_1 (3).wav
Saving [sax][cla]1573_1.wav to [sax][cla]1573_1 (3).wav
Saving [tru][cla]1870_1.wav to [tru][cla]1870_1 (3).wav
Saving [vio][cla]2083_1.wav to [vio][cla]2083_1 (3).wav
Saving [voi][jaz_blu]2334_1.wav to [voi][jaz_blu]2334_1 (3).wav
Files to process (11):
- /content/uploads/[cel][cla]0001_1 (4).wav
- /content/uploads/[cla][cla]0150_1 (3).wav
- /content/uploads/[flu][cla]0346_1 (3).wav
- /content/uploads/[gac][cla]0518_1 (3).wav
- /content/uploads/[gel][cla]0901_1 (3).wav
- /content/uploads/[org][jaz_blu]1039_1 (3).wav
- /content/uploads/[pia][cla]1283_1 (3).wav
- /content/uploads/[sax][cla]1573_1 (3).wav
- /content/uploads/[tru][cla]1870_1 (3).wav
- /content/uploads/[vio][cla]2083_1 (3).wav
- /content/uploads/[voi][jaz_blu]2334_1 (3).wav

```

Step 4: Helper function to compute required representations

```

# Returns: (y, sr, S_db, S_mel_db, mfccs)
# - S_db: STFT magnitude in dB (normal spectrogram)
# - S_mel_db: Mel-spectrogram in dB
# - mfccs: MFCC matrix (n_mfcc x time frames)

```

```

def compute_representations(path, sr=16000, n_fft=2048,
hop_length=512, n_mels=128, n_mfcc=20):
    """
    Load audio and compute:
        - STFT magnitude (converted to dB)
        - Mel-spectrogram (dB)
        - MFCCs (from mel-spectrogram)
    """
    # Load audio
    y, sr = librosa.load(path, sr=sr, mono=True)

    # STFT magnitude spectrogram
    # Note: explicit y=y is safer here too, though strictly required
    for melspectrogram
    S = np.abs(librosa.stft(y=y, n_fft=n_fft, hop_length=hop_length))
    S_db = librosa.amplitude_to_db(S, ref=np.max)

    # Mel spectrogram
    # FIX: Pass 'y' as a keyword argument (y=y)
    S_mel = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=n_fft,
    hop_length=hop_length, n_mels=n_mels)
    S_mel_db = librosa.power_to_db(S_mel, ref=np.max)

    # MFCCs
    # We can pass the already computed log-power Mel spectrogram
    (S_mel_db) to avoid recalculation
    mfccs = librosa.feature.mfcc(S=S_mel_db, n_mfcc=n_mfcc)

    return y, sr, S_db, S_mel_db, mfccs

```

Step 5: Plot function - displays the three plots side-by-side

```

def plot_three_reps(S_db, S_mel_db, mfccs, sr, hop_length=512,
title_prefix=""):
    """
    Plots:
        1) Normal spectrogram (STFT magnitude in dB)
        2) Mel-spectrogram (dB)
        3) MFCC matrix
    """
    plt.figure(figsize=(18, 4))

    # 1. Normal spectrogram
    plt.subplot(1, 3, 1)
    librosa.display.specshow(S_db, sr=sr, hop_length=hop_length,
    x_axis='time', y_axis='linear')
    plt.colorbar(format='%.2f dB')
    plt.title(f"{title_prefix} - STFT Spectrogram (dB)")
    plt.xlabel("Time (s)")

```

```

plt.ylabel("Frequency (Hz)")

# 2. Mel-spectrogram
plt.subplot(1, 3, 2)
librosa.display.specshow(S_mel_db, sr=sr, hop_length=hop_length,
x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title(f"{title_prefix} - Mel-Spectrogram (dB)")
plt.xlabel("Time (s)")
plt.ylabel("Mel frequency")

# 3. MFCC
plt.subplot(1, 3, 3)
librosa.display.specshow(mfccs, sr=sr, hop_length=hop_length,
x_axis='time')
plt.colorbar()
plt.title(f"{title_prefix} - MFCC (n_mfcc x frames)")
plt.xlabel("Time (s)")
plt.ylabel("MFCC coefficient index")

plt.tight_layout()
plt.show()

```

Step 6: Compute and display plots for each uploaded file (up to MAX_FILES)

```

# Combined: Display and Save Spectrograms in One Loop

HOP_LENGTH = 512
save_dir = "/content/figures"
os.makedirs(save_dir, exist_ok=True)

for idx, path in enumerate(filenames, start=1):
    print(f"\nProcessing file {idx}/{len(filenames)}:
{os.path.basename(path)}")

    # Compute features
    y, sr, S_db, S_mel_db, mfccs = compute_representations(
        path,
        sr=16000,                      # <-- Keep consistent
        hop_length=HOP_LENGTH
    )

    # Play the audio
    display(Audio(path))

    # -----
    # 1) DISPLAY FIGURES
    # -----

```

```

plt.figure(figsize=(18, 4))

# STFT Spectrogram
plt.subplot(1, 3, 1)
librosa.display.specshow(S_db, sr=sr, hop_length=HOP_LENGTH,
x_axis='time', y_axis='linear')
plt.colorbar(format='%+2.0f dB')
plt.title("STFT Spectrogram (dB)")
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")

# Mel-Spectrogram
plt.subplot(1, 3, 2)
librosa.display.specshow(S_mel_db, sr=sr, hop_length=HOP_LENGTH,
x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title("Mel-Spectrogram (dB)")
plt.xlabel("Time (s)")
plt.ylabel("Mel Frequency")

# MFCC
plt.subplot(1, 3, 3)
librosa.display.specshow(mfccs, sr=sr, hop_length=HOP_LENGTH,
x_axis='time')
plt.colorbar()
plt.title("MFCC")
plt.xlabel("Time (s)")
plt.ylabel("MFCC Index")

plt.tight_layout()
plt.show()

# -----
# 2) SAVE SAME FIGURE
# -----
# Re-create the same figure for saving
fig = plt.figure(figsize=(18, 4))

# STFT Spectrogram
ax1 = fig.add_subplot(1, 3, 1)
librosa.display.specshow(S_db, sr=sr, hop_length=HOP_LENGTH,
x_axis='time', y_axis='linear')
plt.colorbar(format='%+2.0f dB', ax=ax1)
ax1.set_title("STFT Spectrogram (dB)")
ax1.set_xlabel("Time (s)")
ax1.set_ylabel("Frequency (Hz)")

# Mel-Spectrogram
ax2 = fig.add_subplot(1, 3, 2)
librosa.display.specshow(S_mel_db, sr=sr, hop_length=HOP_LENGTH,

```

```

x_axis='time', y_axis='mel')
plt.colorbar(format='%.+2.0f dB', ax=ax2)
ax2.set_title("Mel-Spectrogram (dB)")
ax2.set_xlabel("Time (s)")
ax2.set_ylabel("Mel Frequency")

# MFCC
ax3 = fig.add_subplot(1, 3, 3)
librosa.display.specshow(mfccs, sr=sr, hop_length=HOP_LENGTH,
x_axis='time')
plt.colorbar(ax=ax3)
ax3.set_title("MFCC")
ax3.set_xlabel("Time (s)")
ax3.set_ylabel("MFCC Index")

fig.tight_layout()

# Save file
out_path = os.path.join(save_dir,
f"{os.path.splitext(os.path.basename(path))[0]}_reprs.png")
fig.savefig(out_path, dpi=200)
plt.close(fig)

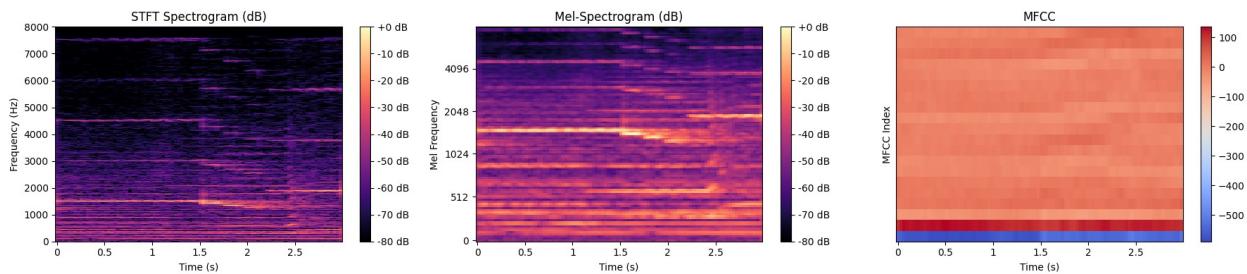
print(f"Saved: {out_path}")

print("\nAll figures displayed and saved to:", save_dir)

```

Processing file 1/11: [cel][cla]0001_1 (4).wav

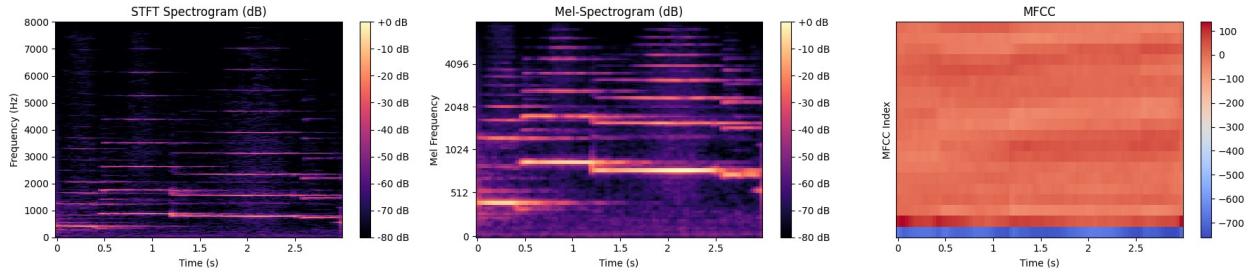
<IPython.lib.display.Audio object>



Saved: /content/figures/[cel][cla]0001_1 (4)_reprs.png

Processing file 2/11: [cla][cla]0150_1 (3).wav

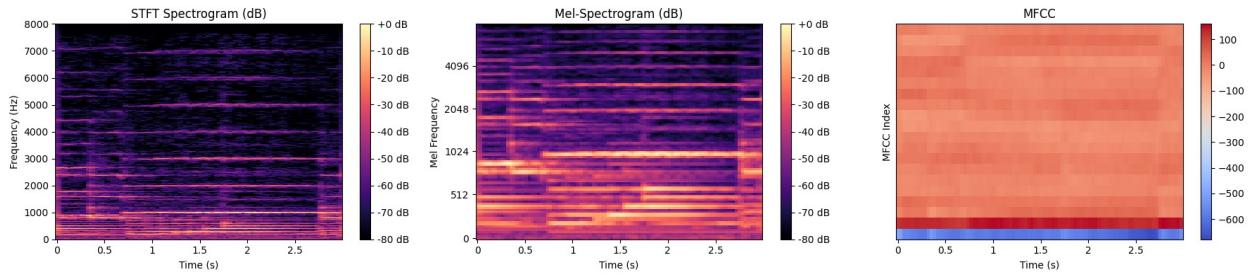
<IPython.lib.display.Audio object>



Saved: /content/figures/[cla][cla]0150_1 (3)_reprs.png

Processing file 3/11: [flu][cla]0346_1 (3).wav

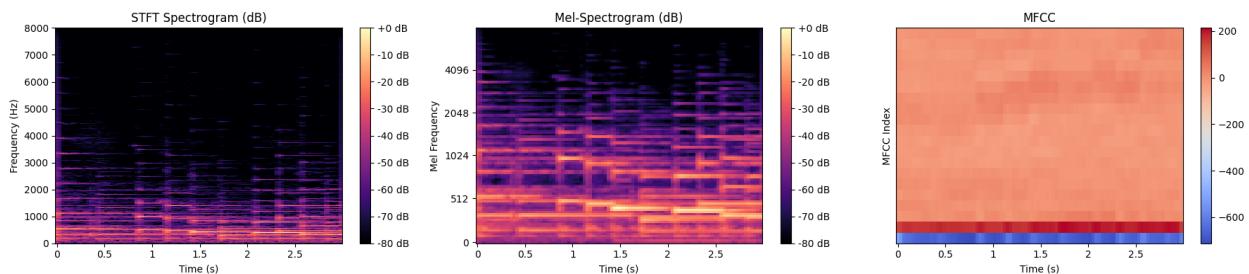
<IPython.lib.display.Audio object>



Saved: /content/figures/[flu][cla]0346_1 (3)_reprs.png

Processing file 4/11: [gac][cla]0518_1 (3).wav

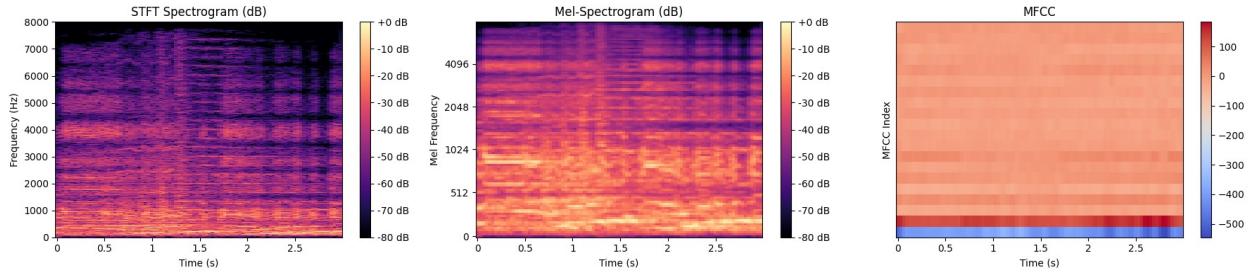
<IPython.lib.display.Audio object>



Saved: /content/figures/[gac][cla]0518_1 (3)_reprs.png

Processing file 5/11: [gel][cla]0901_1 (3).wav

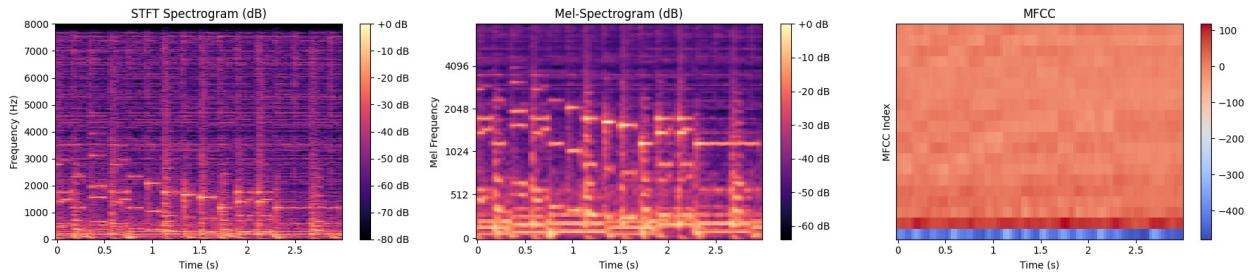
<IPython.lib.display.Audio object>



Saved: /content/figures/[gel][cla]0901_1 (3)_reprs.png

Processing file 6/11: [org][jaz_blu]1039_1 (3).wav

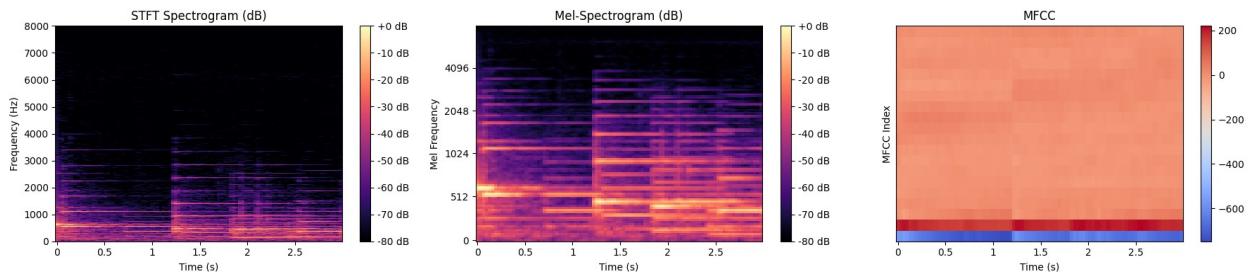
<IPython.lib.display.Audio object>



Saved: /content/figures/[org][jaz_blu]1039_1 (3)_reprs.png

Processing file 7/11: [pia][cla]1283_1 (3).wav

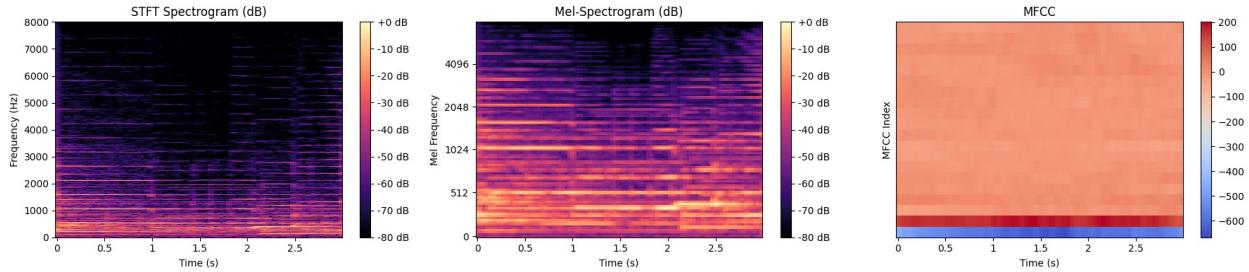
<IPython.lib.display.Audio object>



Saved: /content/figures/[pia][cla]1283_1 (3)_reprs.png

Processing file 8/11: [sax][cla]1573_1 (3).wav

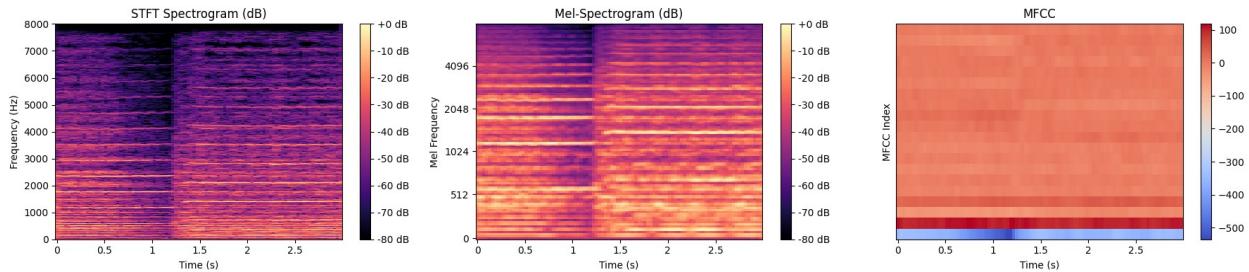
<IPython.lib.display.Audio object>



Saved: /content/figures/[sax][cla]1573_1 (3)_reprs.png

Processing file 9/11: [tru][cla]1870_1 (3).wav

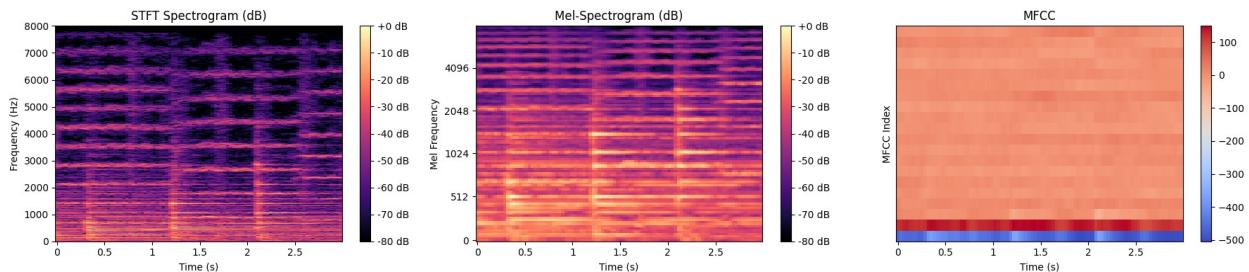
<IPython.lib.display.Audio object>



Saved: /content/figures/[tru][cla]1870_1 (3)_reprs.png

Processing file 10/11: [vio][cla]2083_1 (3).wav

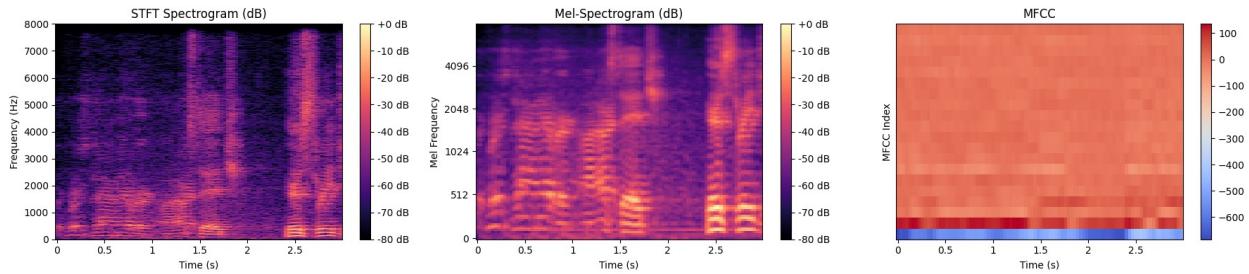
<IPython.lib.display.Audio object>



Saved: /content/figures/[vio][cla]2083_1 (3)_reprs.png

Processing file 11/11: [voi][jaz_blu]2334_1 (3).wav

<IPython.lib.display.Audio object>



```
Saved: /content/figures/[voi][jaz_blu]2334_1 (3)_reprs.png
```

```
All figures displayed and saved to: /content/figures
```

Conclusion:

- (a) The normal spectrogram looks detailed but also quite cluttered and it doesn't appear to be the best choice for a CNN to learn from.
- (b) The mel-spectrogram looks much cleaner compared to STFT spectrogram focusing on the parts of the sound that appear to matter most for recognizing instruments.
- (c) MFCCs look very compressed and appear to lose the visual patterns that CNNs usually depend on.