



Project - Signal Processing Music Genre Classification

Outline

- Giới thiệu – Bộ dữ liệu GTZAN
- Phương pháp tiếp cận
- Xử lý tín hiệu – Nền tảng
- Music Genre Classification

Giới thiệu – Bộ dữ liệu GTZAN

GTZAN Dataset thường
được xem như là bộ
“MNIST”
cho âm nhạc

1000 bản nhạc

10 thể loại

30s/bài

GTZAN Dataset - Music Genre Classification

Audio Files | Mel Spectrograms | CSV with extracted features

Data Card Code (172) Discussion (17)



About Dataset

Context

Music. Experts have been trying for a long time to understand sound and what differentiates one song from another. How to visualize sound. What makes a tone different from another.

This data hopefully can give the opportunity to do just that.

Content

- **genres original** - A collection of 10 genres with 100 audio files each, all having a length of 30 seconds (the famous GTZAN dataset, the MNIST of sounds)
- **images original** - A visual representation for each audio file. One way to classify data is through neural networks. Because NNs (like CNN, what we will be using today) usually take in some sort of image representation, the audio files were converted to Mel Spectrograms to make this possible.
- **2 CSV files** - Containing features of the audio files. One file has for each song (30 seconds long) a mean and variance computed over multiple features that can be extracted from an audio file. The other file has the same structure, but the songs were split before into 3 seconds audio files (this way increasing 10 times the amount of data we fuel into our classification models). *With data, more is always better.*

Usability

8.82

License

Other (specified in description)

Expected update frequency

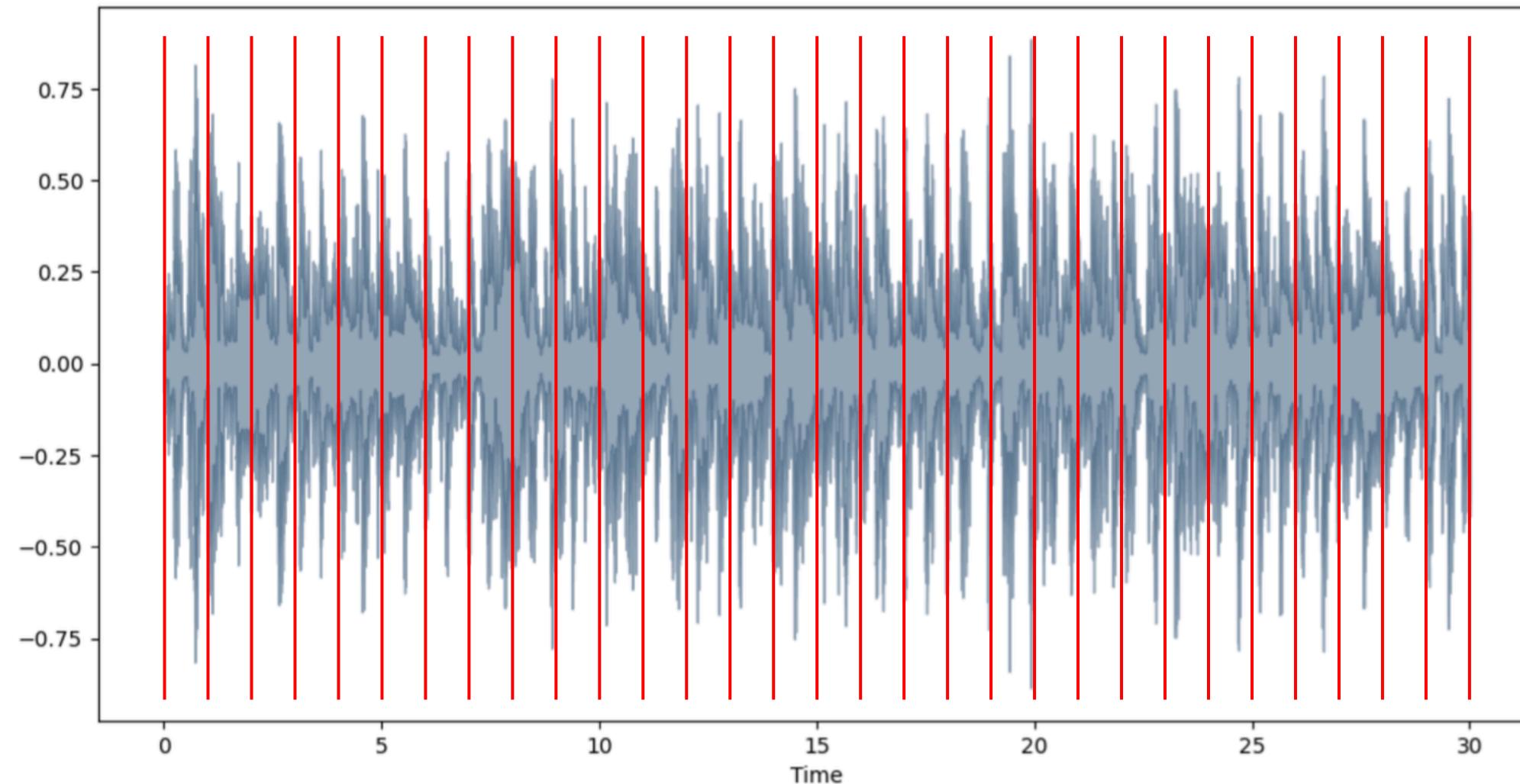
Never

Tags

Music

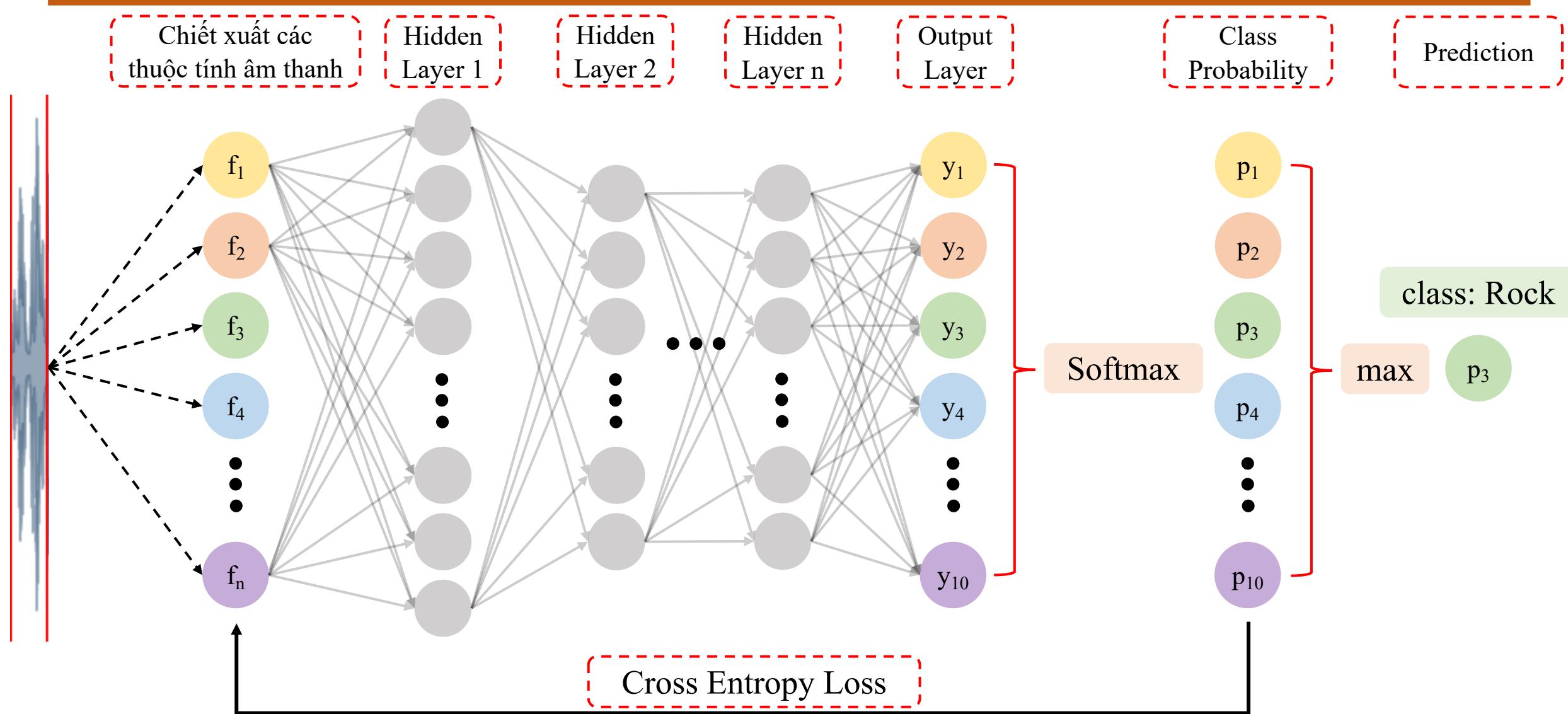
Feature Engineering

Phương pháp tiếp cận

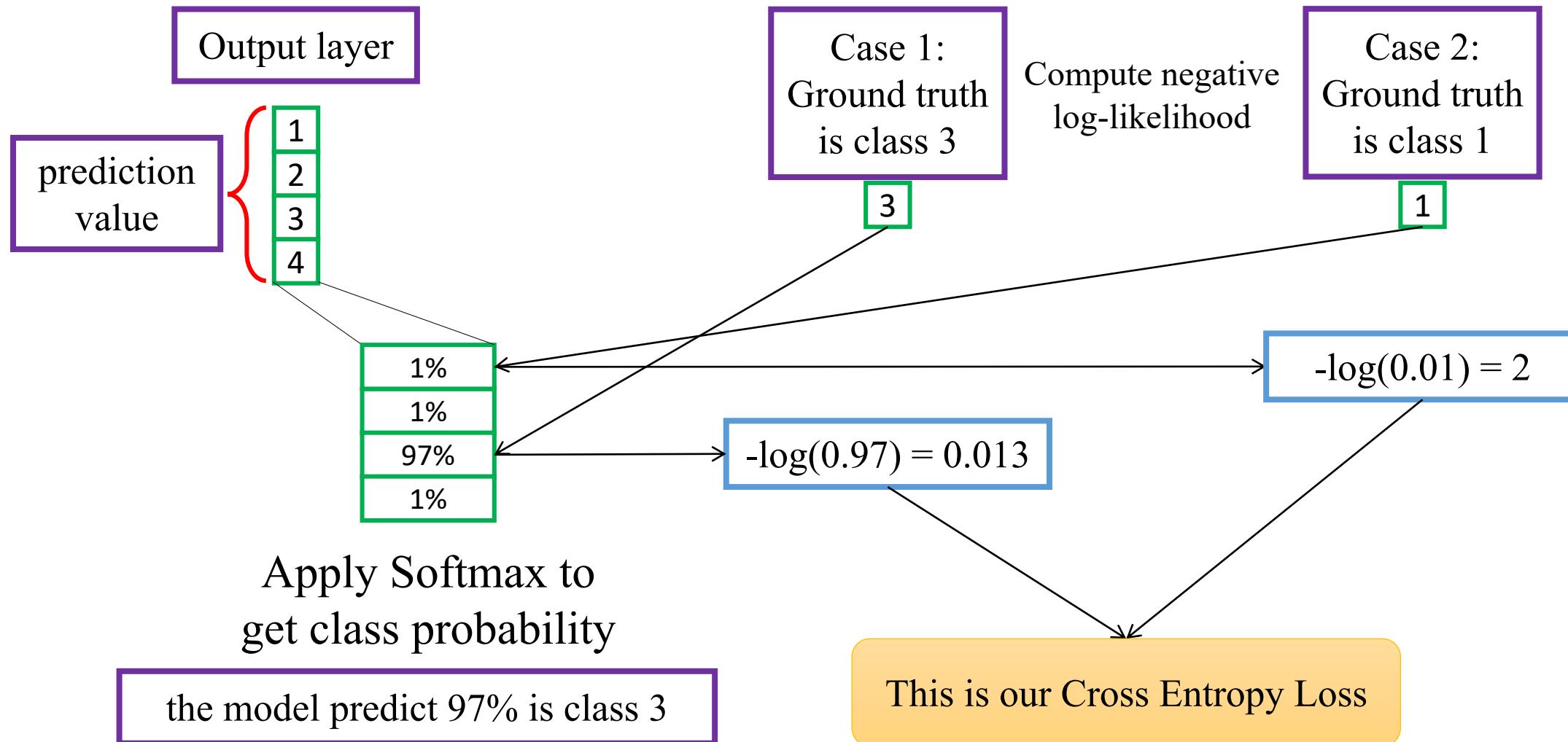


Chúng ta sẽ tăng
số lượng data
bằng cách phân
mỗi bản nhạc
thành 10 đoạn
nhỏ 3s

Phương pháp tiếp cận



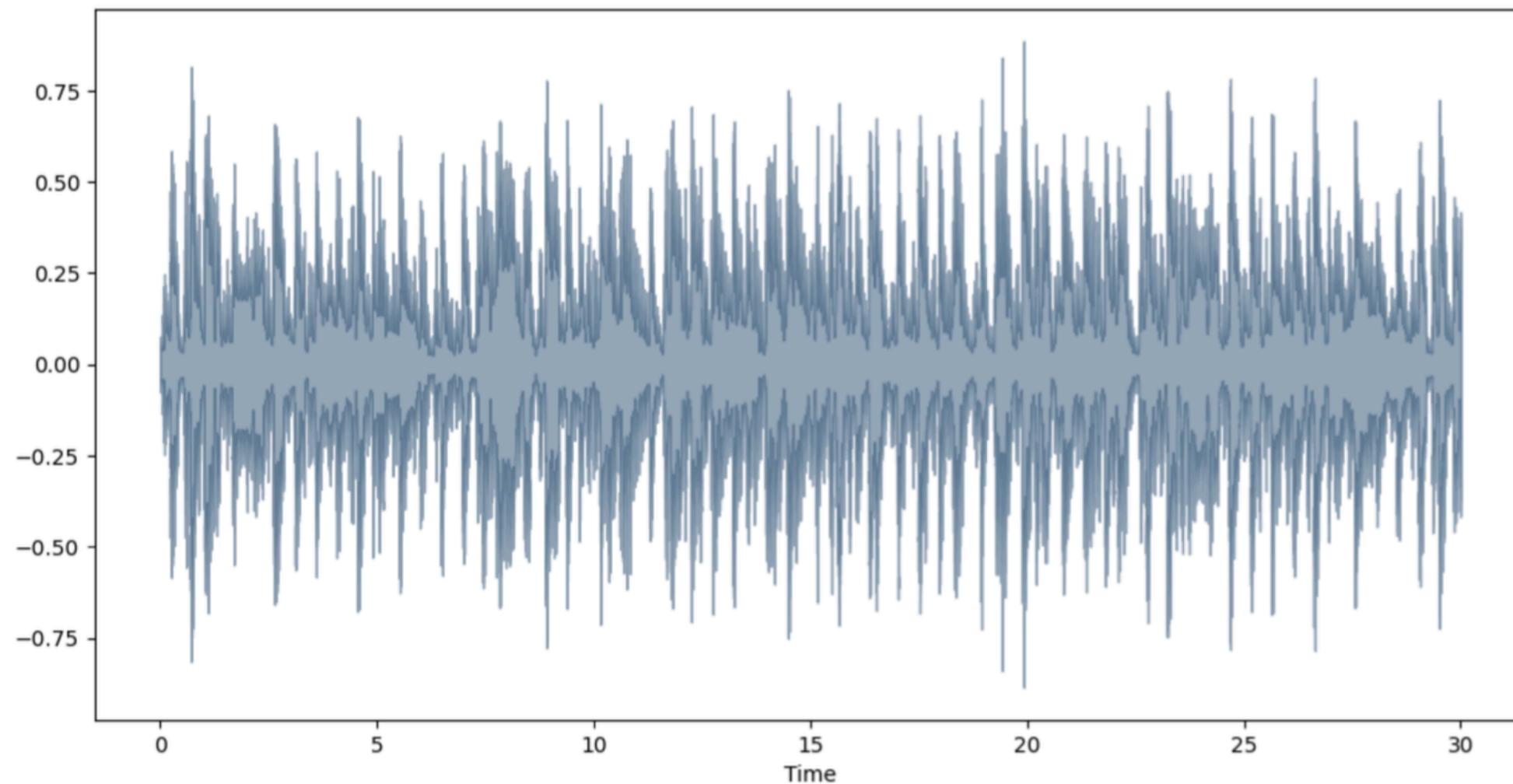
Cross Entropy Loss





Xử lý tín hiệu Nền tảng

Biểu đồ sóng (Waveform)



Biểu diễn hình ảnh của
một tín hiệu âm thanh

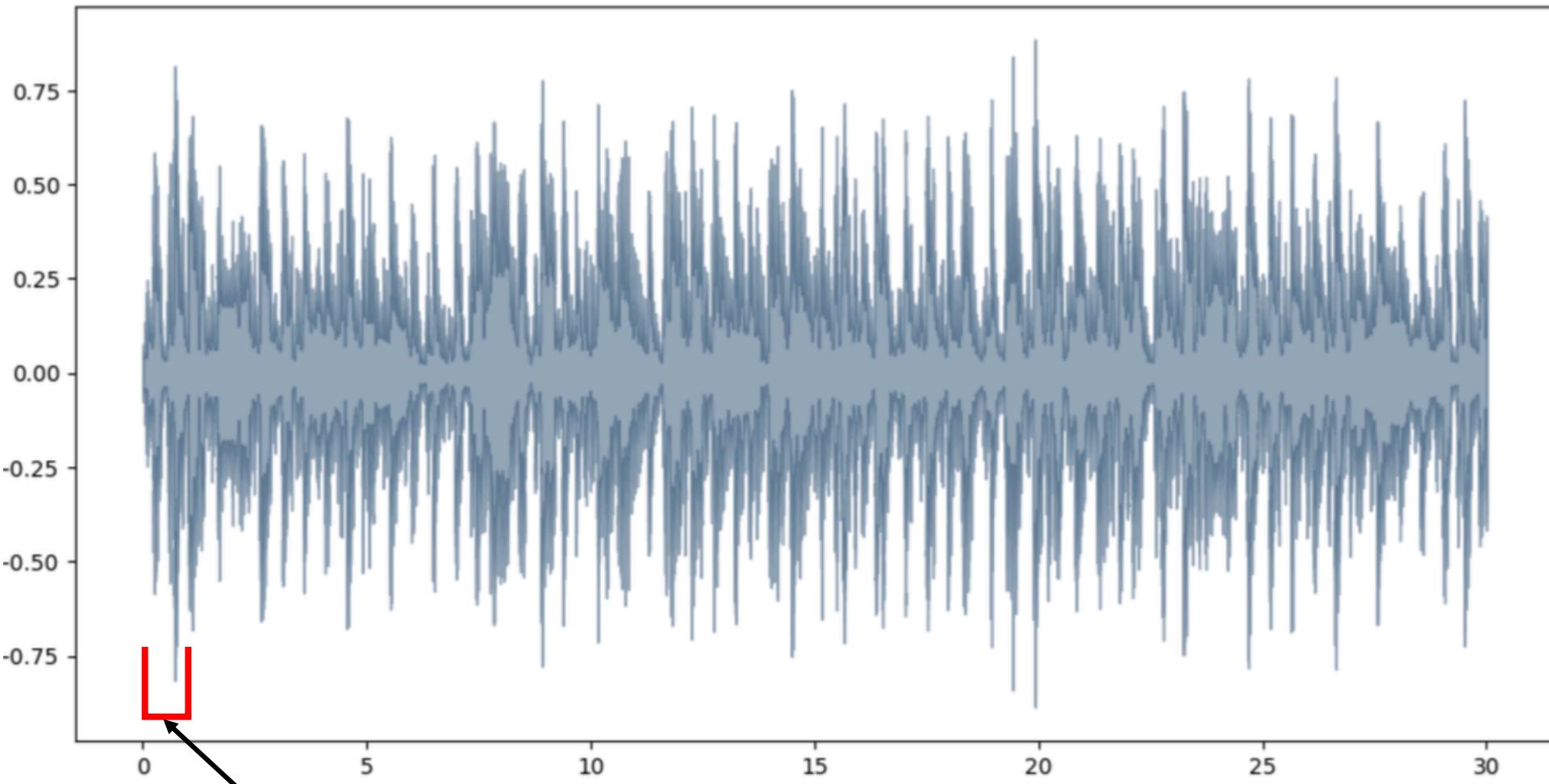
Hiển thị biên độ tín
hiệu theo thời gian

Trục dọc biểu diễn
cường độ của âm
thanh.

Trục ngang biểu diễn
thời gian

Hình 1: Biểu đồ sóng - Waveform

Tốc độ lấy mẫu (Sampling Rate)



Âm thanh là một tín hiệu liên tục

Máy tính cần tín hiệu rời rạc để xử lý

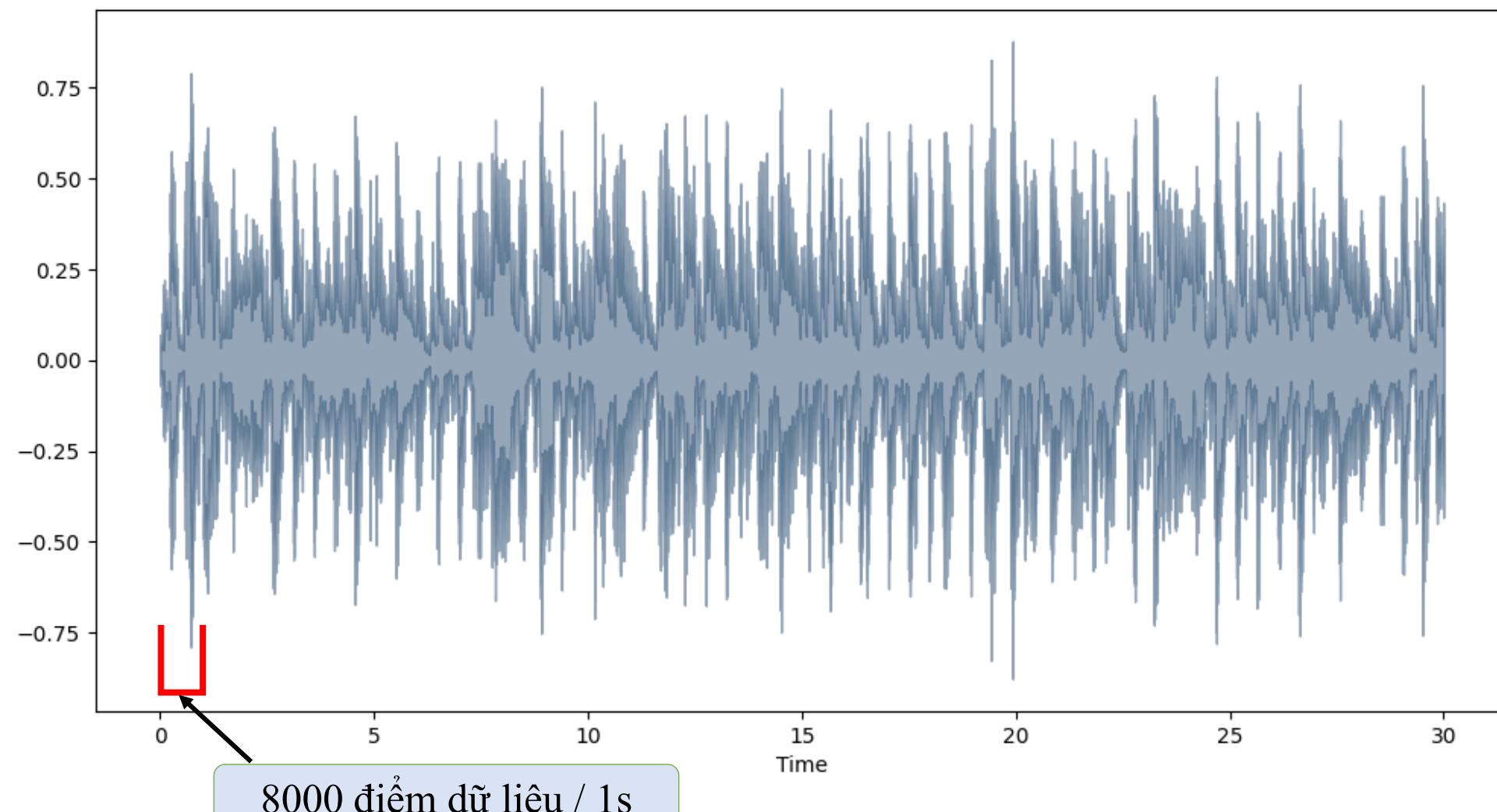
Sampling rate =
Số lượng điểm dữ liệu
lấy trong 1 giây

**Nyquist-Shannon
Sampling Theorem**

MIR thường chọn
 $sr = 22.050\text{Hz}$
 $sr = 44.100\text{Hz}$

Hình 2: Biểu đồ sóng - Sampling rate = 22.050 Hz

Tốc độ lấy mẫu (Sampling Rate)



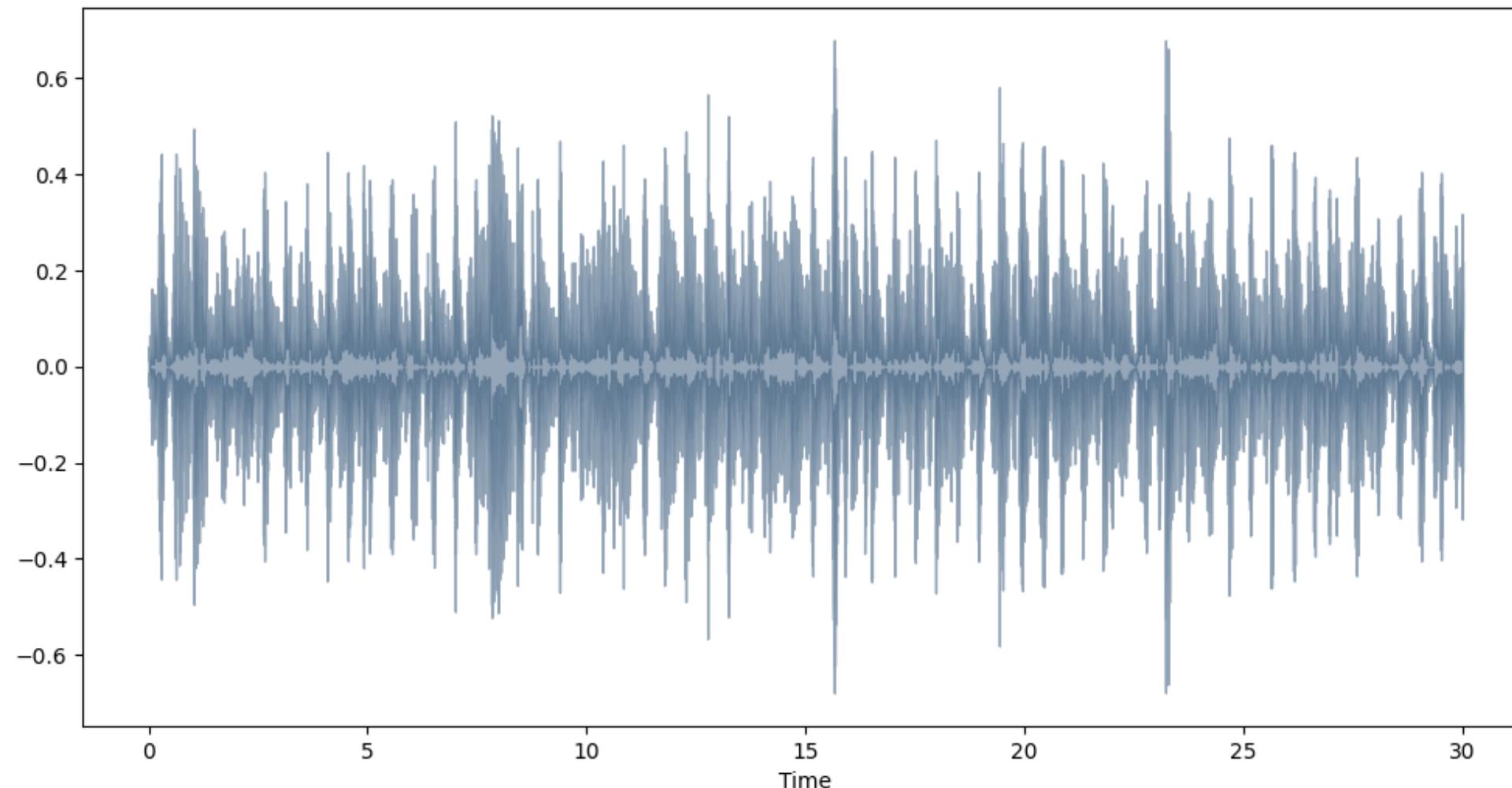
SR càng lớn càng thể hiện chính xác tín hiệu

Tuy nhiên số lượng cần xử lý rất lớn

Thông thường trong xử lý giọng nói sr phù hợp
 $sr = 8000 \text{ Hz}$
 $sr = 16000 \text{ Hz}$

Hình 3: Biểu đồ sóng - Sampling rate = 8000 Hz

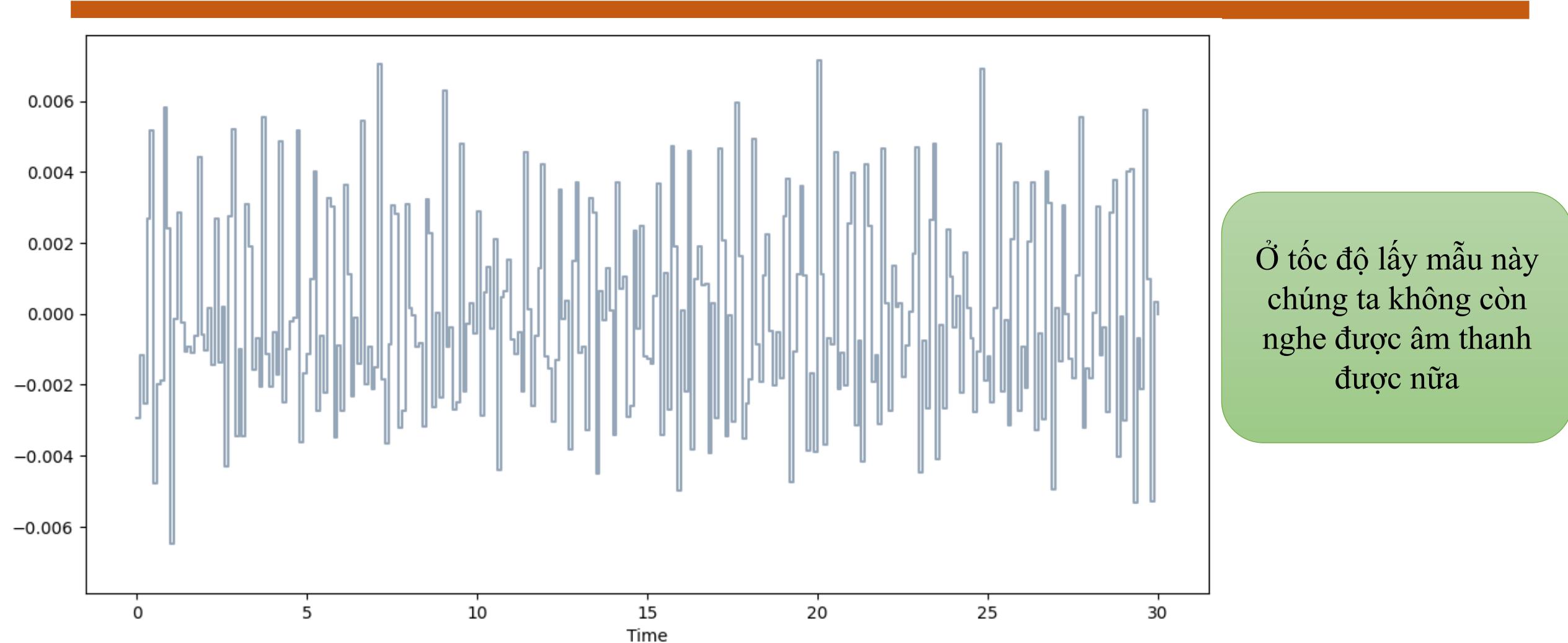
Tốc độ lấy mẫu (Sampling Rate)



Chúng ta gần như
không thể nhận diện
được bài nhạc ở tốc độ
lấy mẫu này

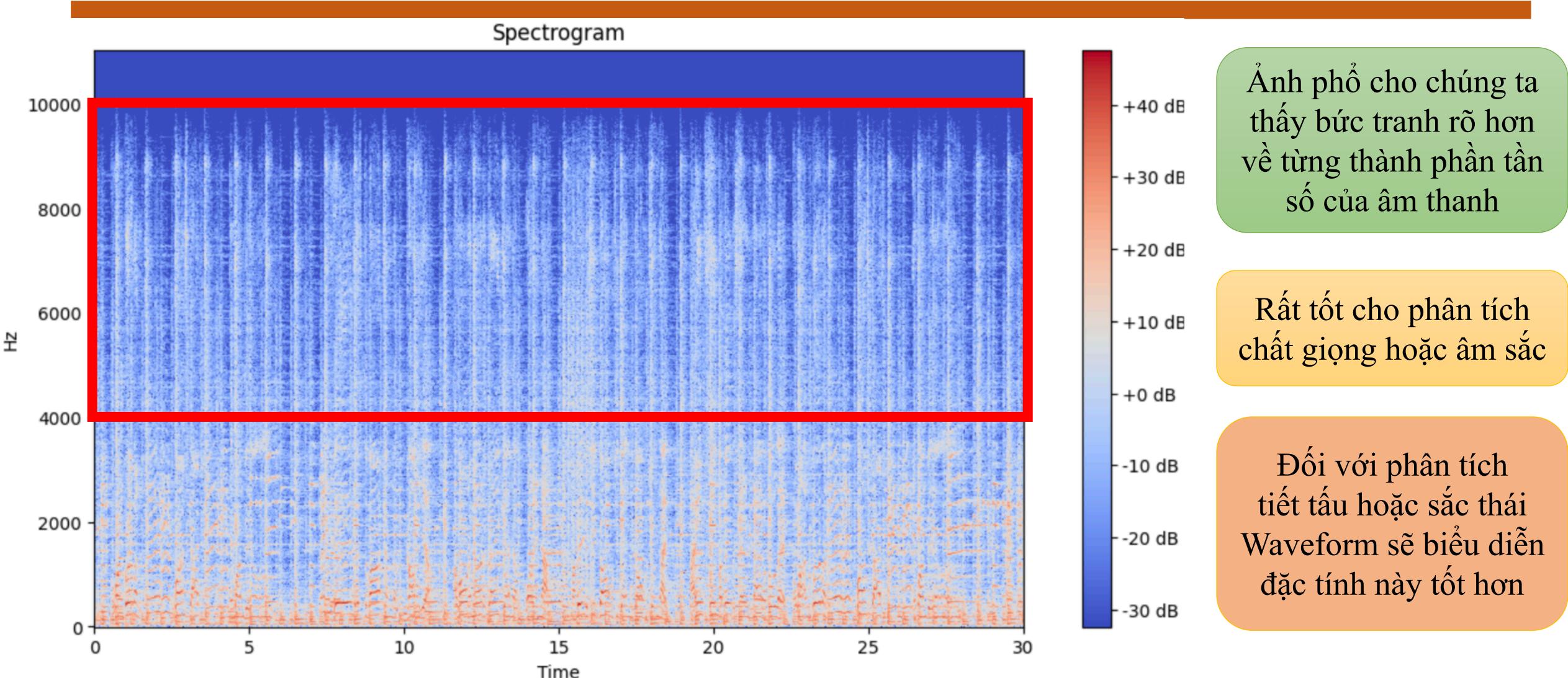
Hình 4: Biểu đồ sóng - Sampling rate = 1000 Hz

Tốc độ lấy mẫu (Sampling Rate)



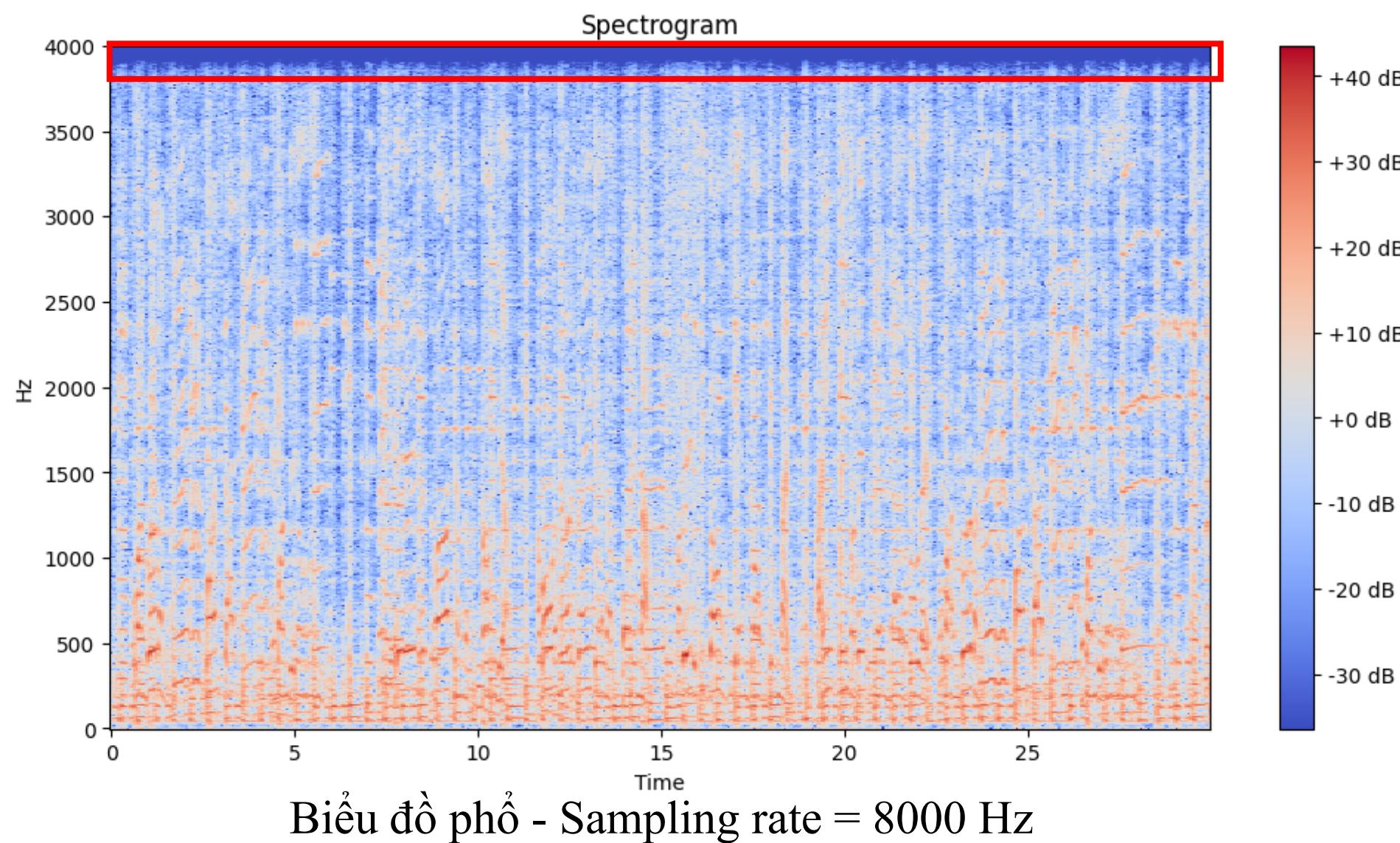
Hình 5: Biểu đồ sóng - Sampling rate = 10 Hz

Biểu đồ phổ (Spectrogram)



Hình 6: Biểu đồ phổ - Sampling rate = 22.050 Hz

Biểu đồ phổ (Spectrogram)

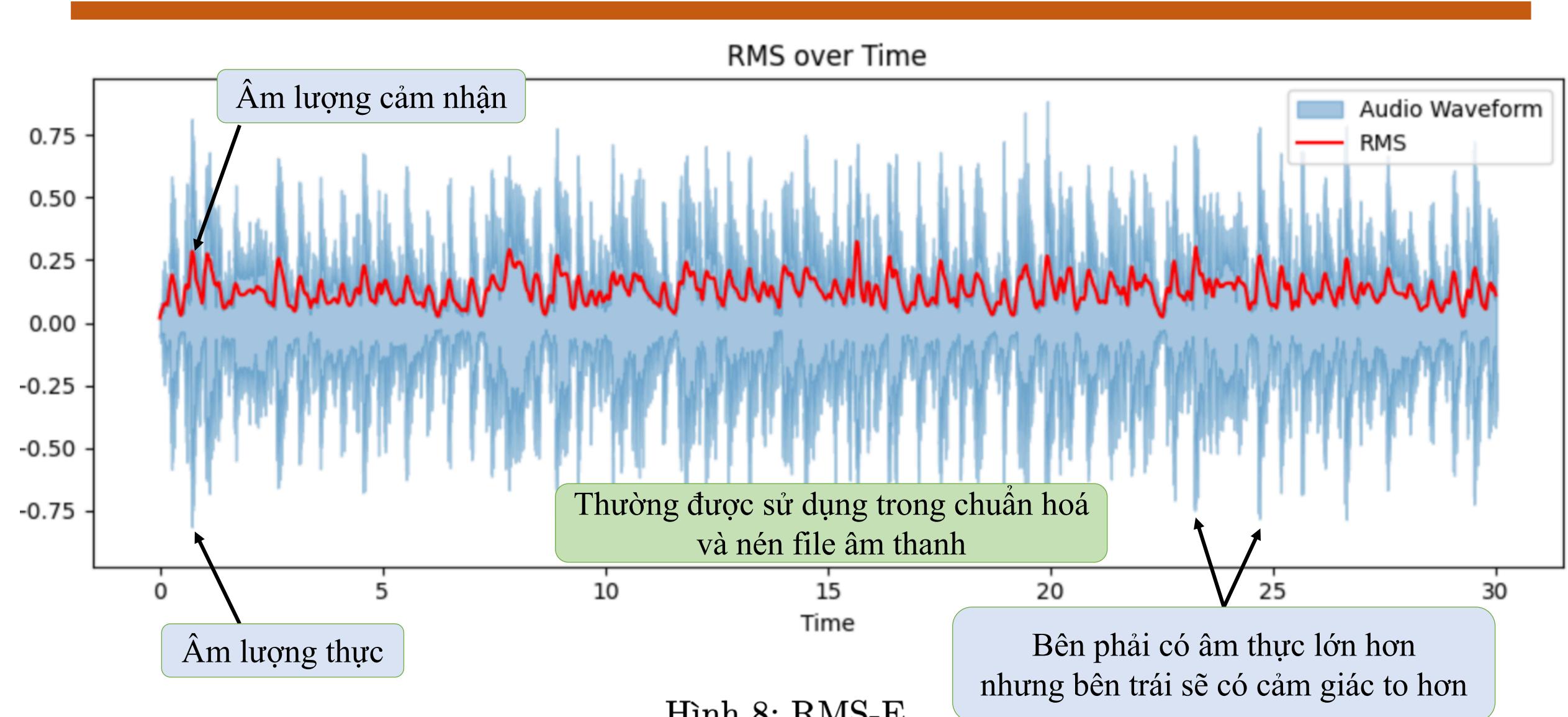


Chúng ta có thể thấy với $sr = 8000$, các vùng tần số $> 4000\text{Hz}$ đã biến mất

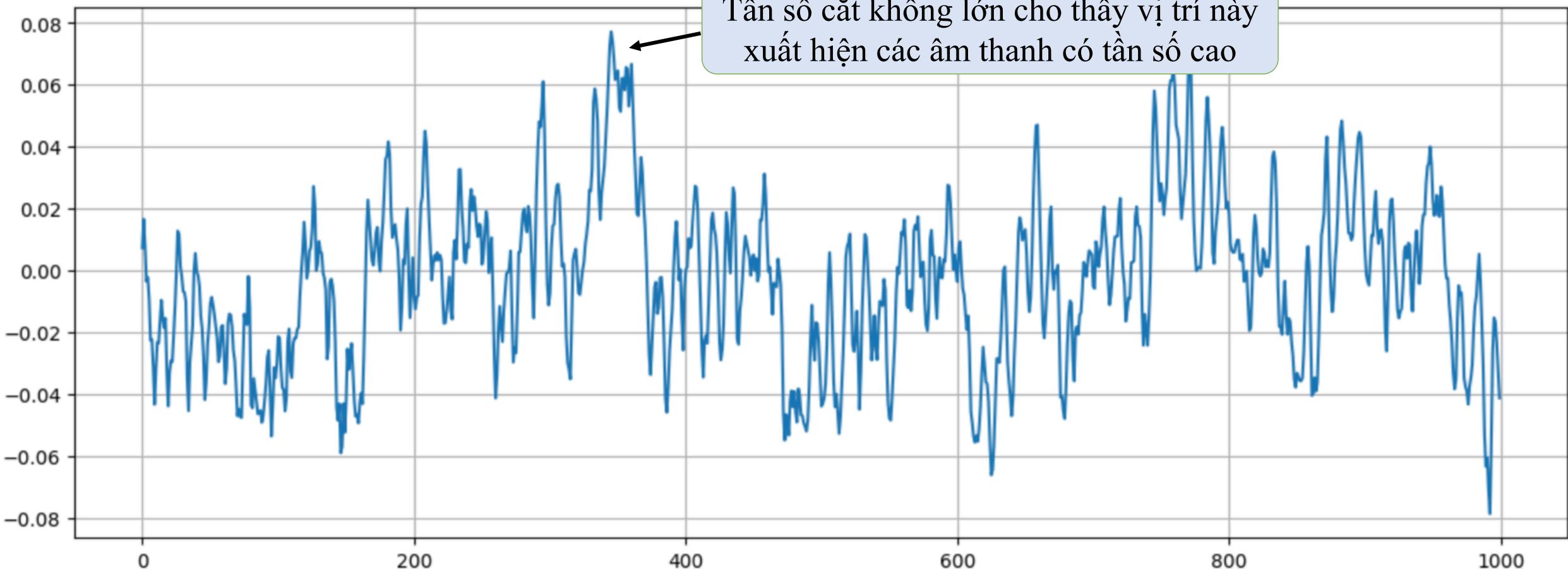
Điều này tuân theo **Nyquist-Shannon Sampling Theorem**

Khi nghe bài nhạc này ở $sr = 8000\text{Hz}$, chúng ta sẽ nghe được sự thiếu hụt những âm thanh cao tầng $> 4000 \text{ Hz}$

Năng lượng hiệu dụng (RMS-E)



Tần số cắt không (Zero-Crossing Rate)

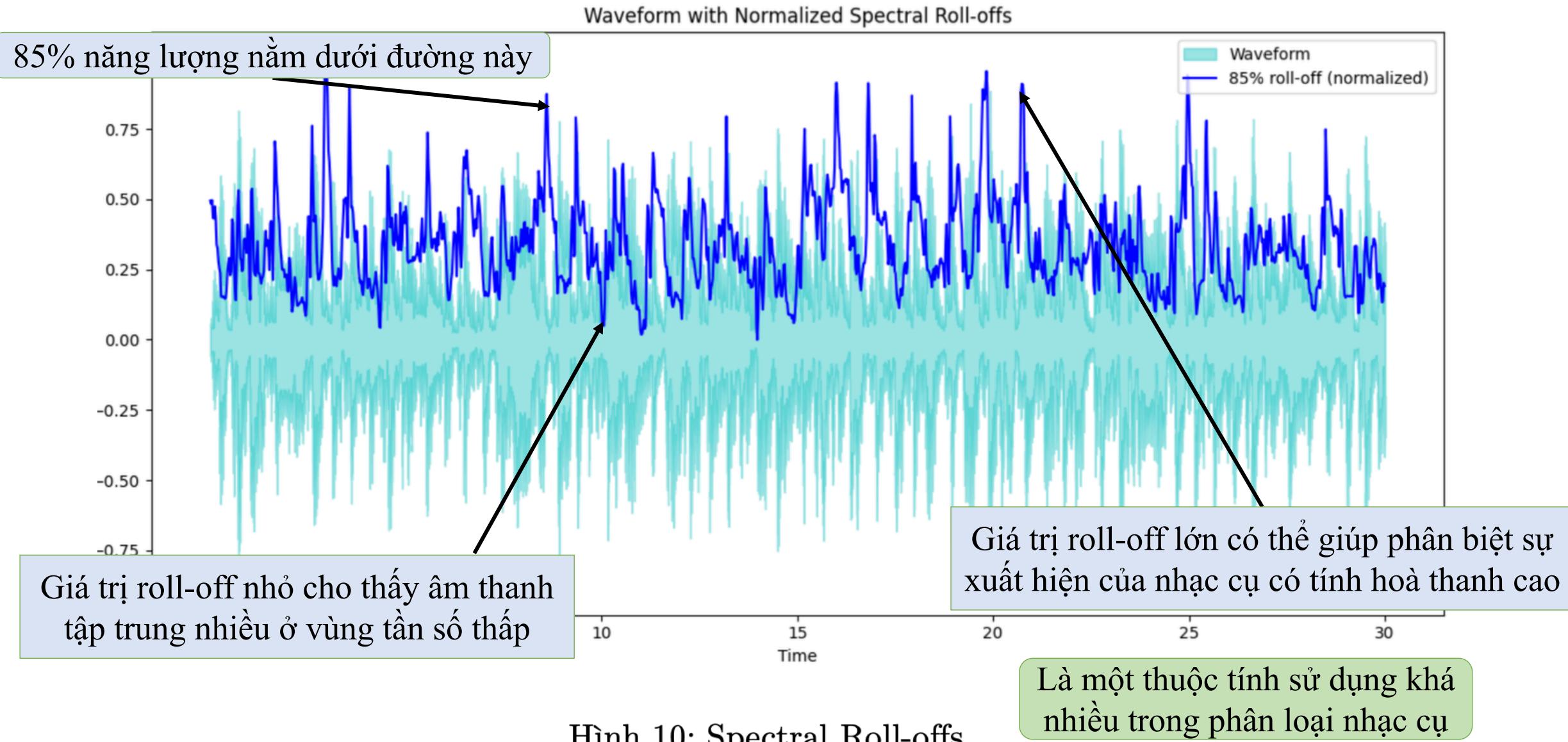


Hình 9: Tần số cắt không

ZCR tăng đột ngột có thể cho thấy sự xuất hiện của Nhiều

Phân tích giọng nói
nguyên âm thường có ZCR thấp
còn phụ âm thường có ZCR cao hơn

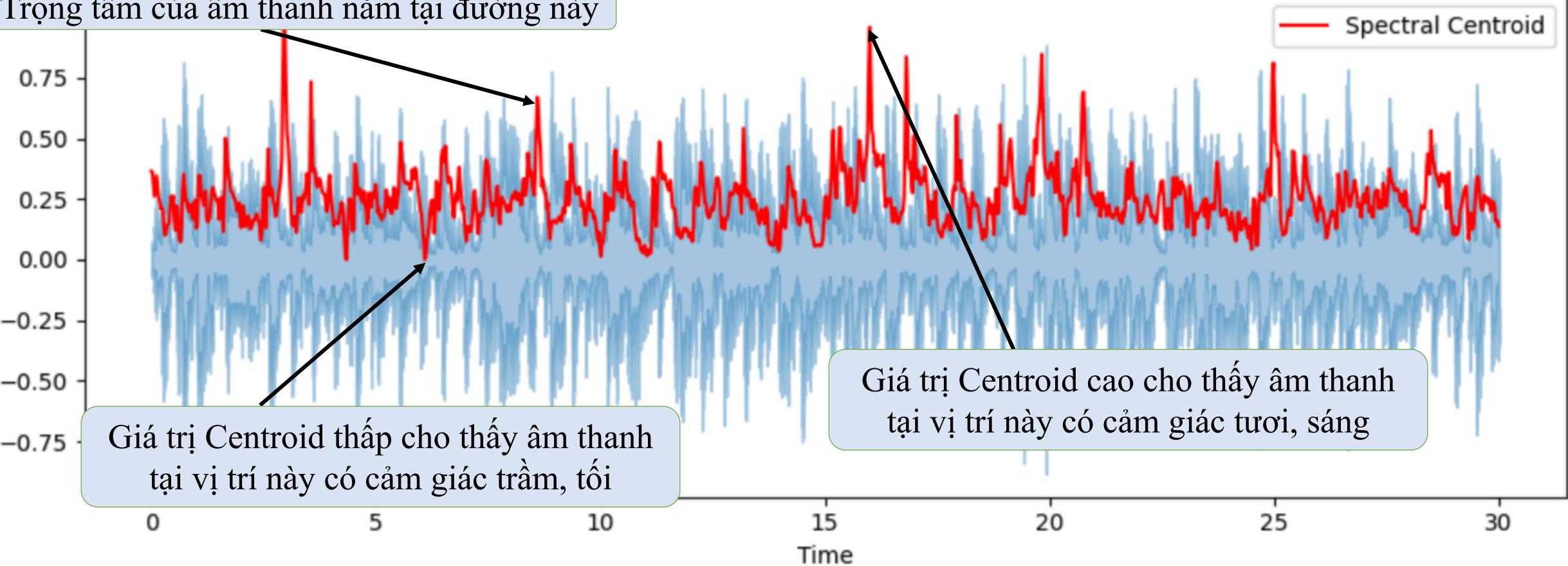
Spectral roll-off



Spectral Centroid

Trọng tâm của âm thanh nằm tại đường này

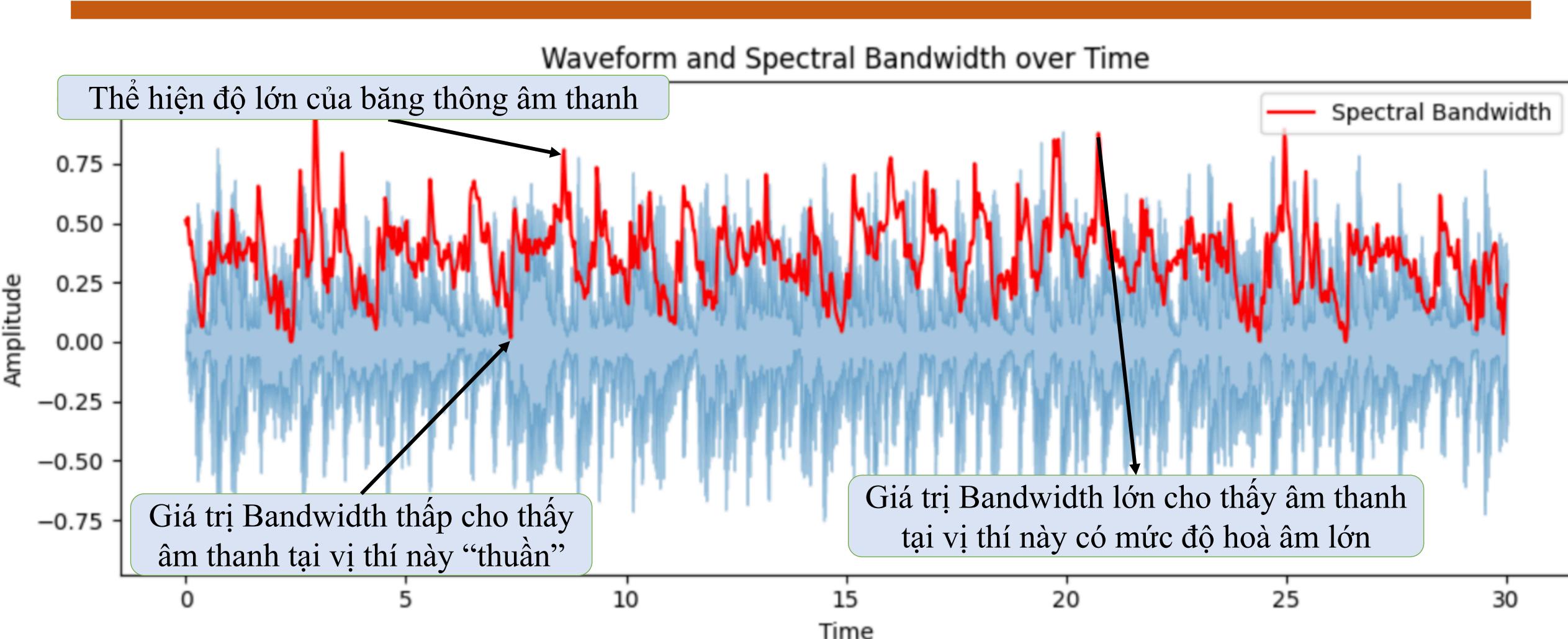
Waveform and Spectral Centroid over Time



Hình 11: Spectral Centroid

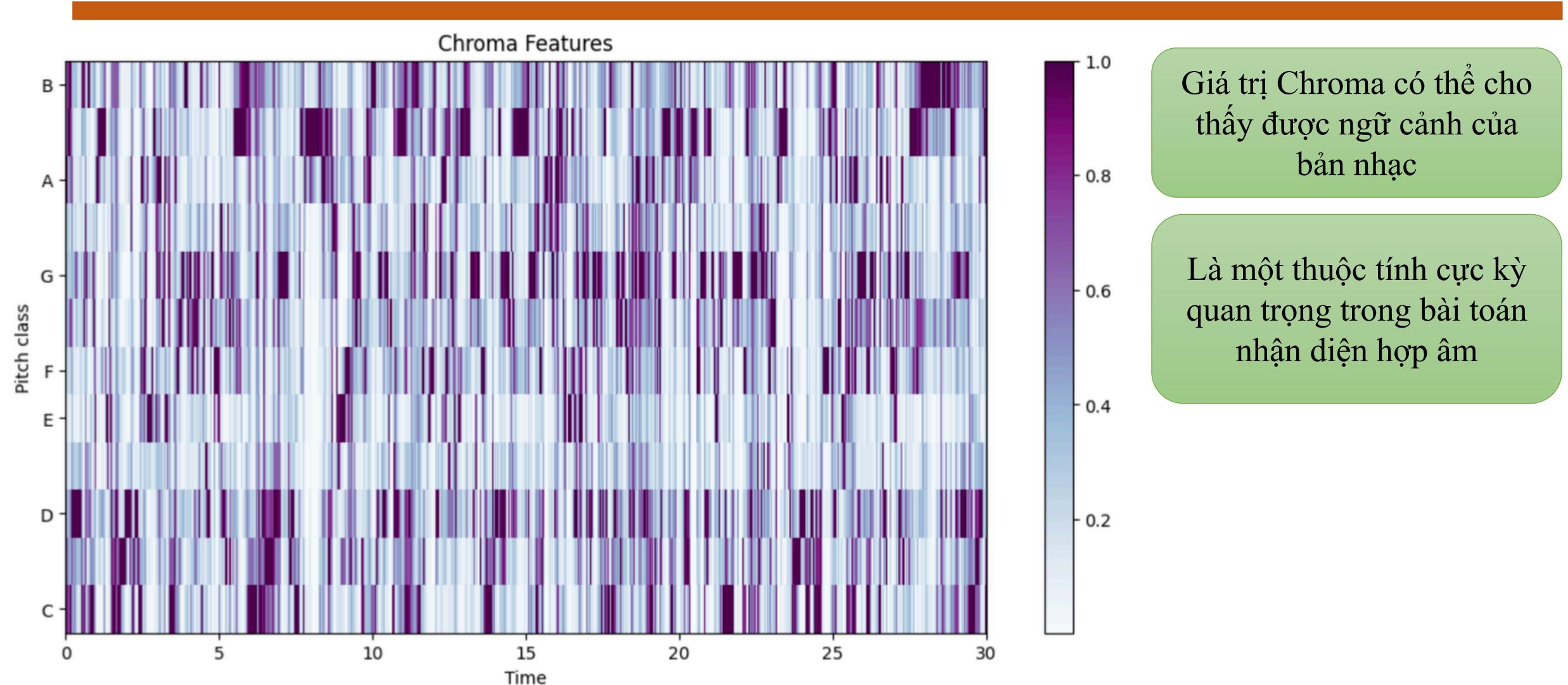
Cho thấy được độ “tươi”/ “sáng” của âm thanh

Spectral Bandwidth



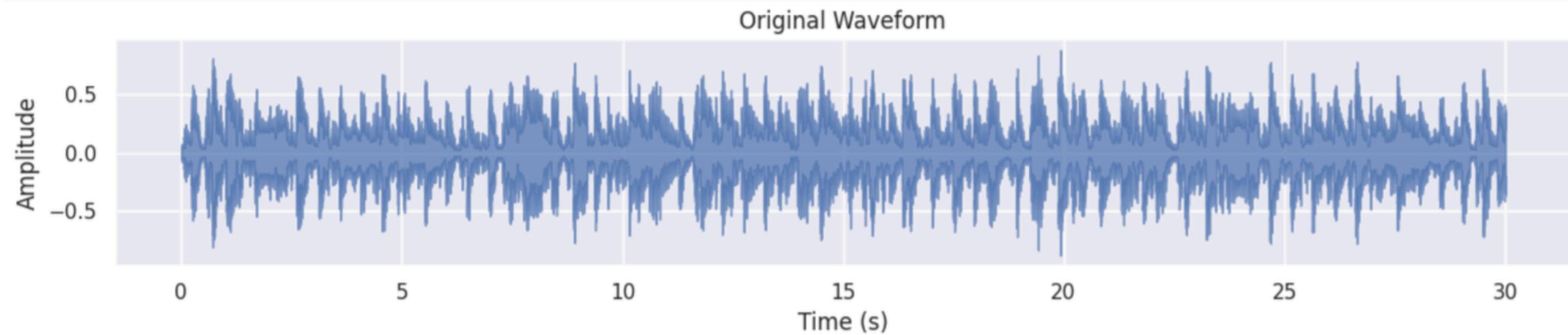
Hình 12: Spectral Bandwidth

Chroma

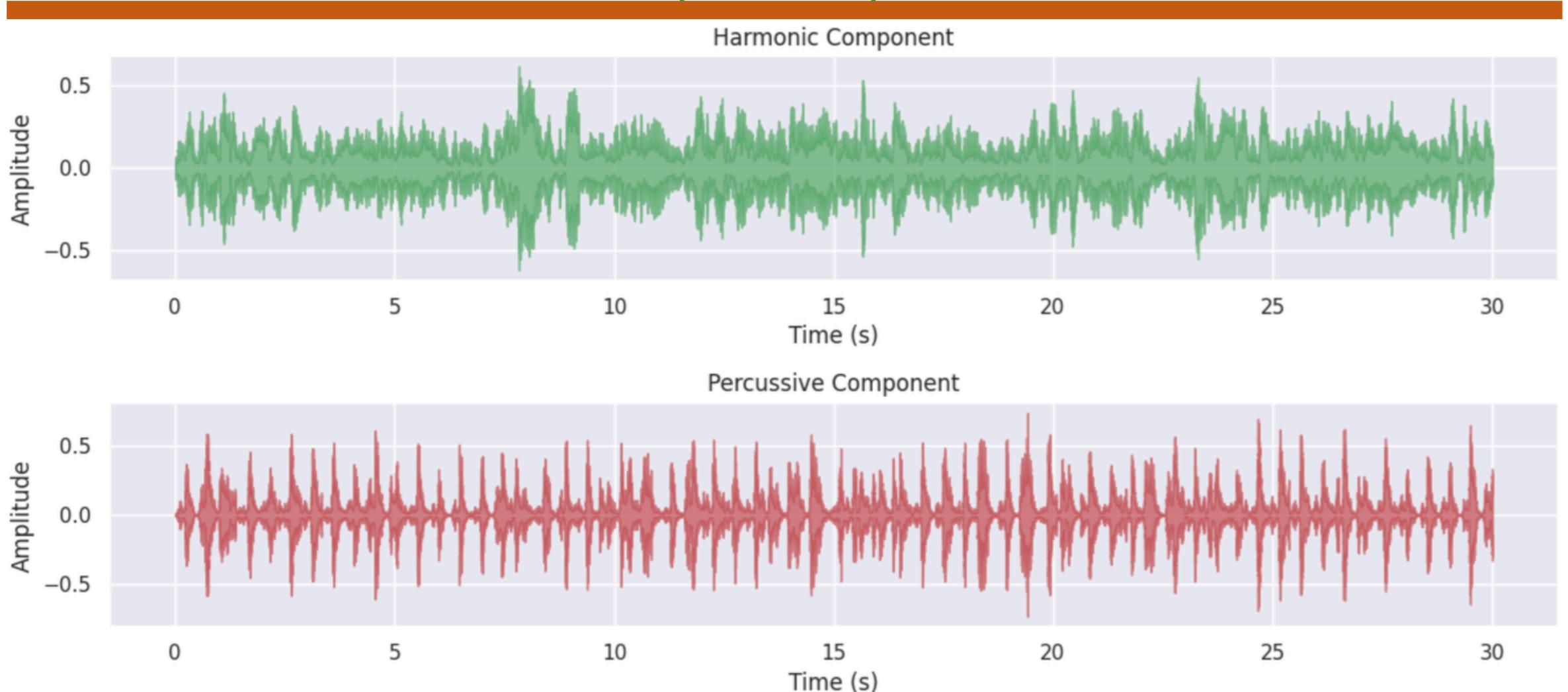


Hình 13: Chroma

Harmonic/Percussive Source Separation (HPSS)

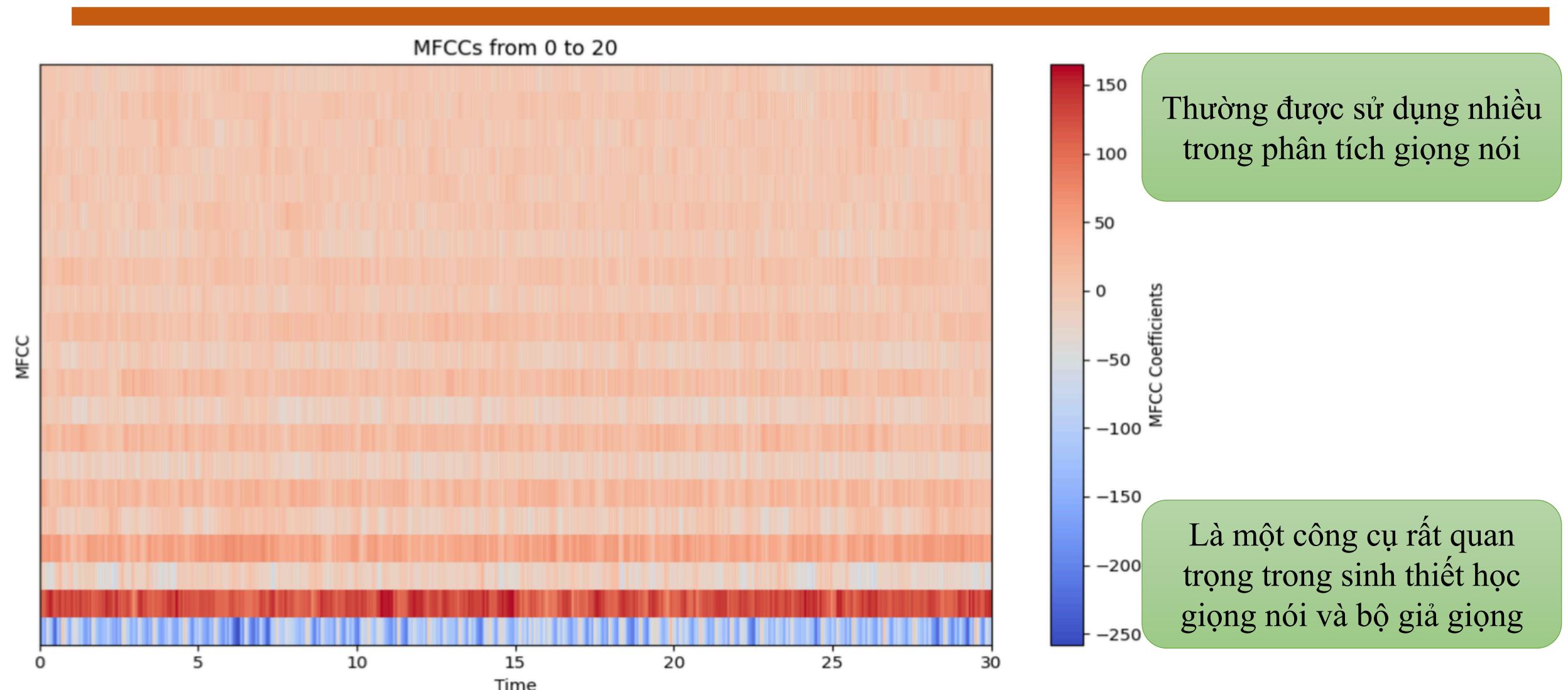


Trong phân tích âm nhạc chúng ta có thể phân tách âm thanh thành 2 thành phần chủ đạo
HOÀ THANH và BỘ GÕ



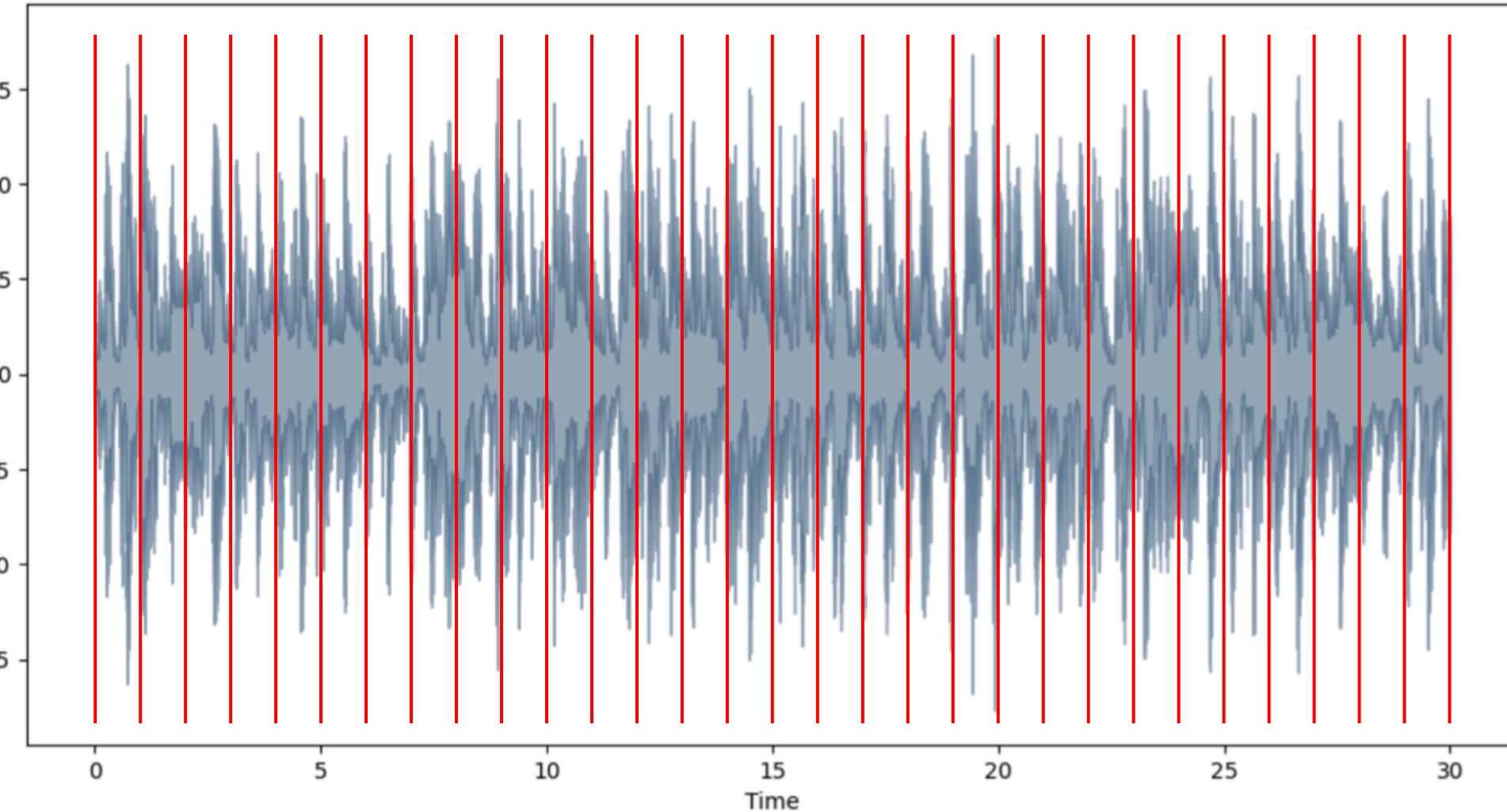
Hình 14: Harmonic/Percussive Source Separation

Hệ số Cepstral Tần số Mel (MFCC)



Hình 15: Hệ số Cepstral Tần số Mel

Music Genre Classification - Project



Phân mői bản nhạc
thành 10 đoạn 3s

Trích xuất và tính
tổn mean và var
của tất cả các
thuộc tính cho các
đoạn nhạc

Chúng ta sẽ có tổng
cộng 60 features cho
mỗi đoạn nhạc

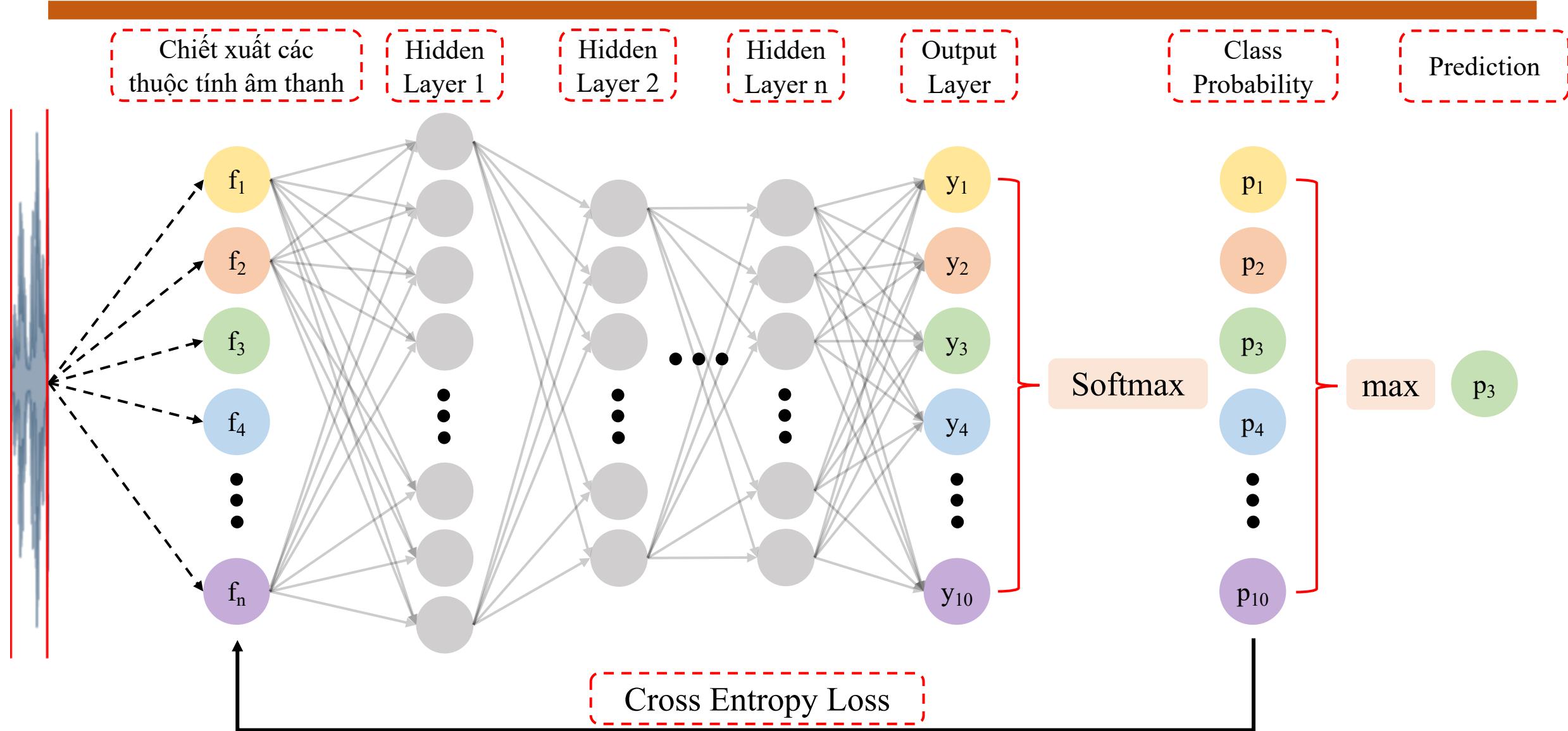
Music Genre Classification - Project

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1788.539719
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1655.289045
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1630.656199

Sau khi phân đoạn và chiết xuất, tính toán các thuộc tính. Chúng ta sẽ có bộ dữ liệu gồm 10000 đoạn nhạc với 60 thuộc tính đã được tính toán. Chúng ta sẽ lưu trữ thông tin này trong file CSV và sử dụng để huấn luyện mô hình.

Chúng ta sẽ sử dụng mô hình Feed Forward Neural Network đơn giản cho bài toán phân loại này với hàm loss Cross Entropy Loss và Adam Optimizer.

Music Genre Classification - Project



Music Genre Classification - Project

```
class MLP(nn.Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(input_size, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 32)
        self.fc6 = nn.Linear(32, 10)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = F.relu(self.fc3(x))
        x = self.dropout(x)
        x = F.relu(self.fc4(x))
        x = self.dropout(x)
        x = F.relu(self.fc5(x))
        x = self.dropout(x)
        x = F.softmax(self.fc6(x), dim=1)
        return x
```

```
input_size = X_train.shape[1]
model = MLP(input_size)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.000146)
```

```
num_epochs = 300
batch_size = 256

train_dataset = TensorDataset(torch.tensor(X_train), torch.tensor(y_train))
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

val_dataset = TensorDataset(torch.tensor(X_test), torch.tensor(y_test))
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

Music Genre Classification - Project

```
step = 0

for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs.float())
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if step % 100 ==0:
            print(f"Step {step}, Train Loss: {loss.item():.4f}")
        step += 1

# Validation
model.eval()
val_loss = 0.0
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in val_loader:
        outputs = model(inputs.float())
        loss = criterion(outputs, labels)
        val_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

val_loss /= len(val_loader)
val_accuracy = 100 * correct / total
print(f"Epoch {epoch+1}/{num_epochs}, Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.2f}%")
```

```
Step 7700, Train Loss: 1.5527
Epoch 276/300, Validation Loss: 1.6205, Validation Accuracy: 83.98%
Epoch 277/300, Validation Loss: 1.6207, Validation Accuracy: 84.18%
Epoch 278/300, Validation Loss: 1.6223, Validation Accuracy: 83.88%
Step 7800, Train Loss: 1.5503
Epoch 279/300, Validation Loss: 1.6233, Validation Accuracy: 83.65%
Epoch 280/300, Validation Loss: 1.6208, Validation Accuracy: 84.02%
Epoch 281/300, Validation Loss: 1.6205, Validation Accuracy: 84.22%
Epoch 282/300, Validation Loss: 1.6217, Validation Accuracy: 83.88%
Step 7900, Train Loss: 1.5556
Epoch 283/300, Validation Loss: 1.6209, Validation Accuracy: 84.22%
Epoch 284/300, Validation Loss: 1.6196, Validation Accuracy: 84.05%
Epoch 285/300, Validation Loss: 1.6190, Validation Accuracy: 84.12%
Step 8000, Train Loss: 1.5322
Epoch 286/300, Validation Loss: 1.6185, Validation Accuracy: 84.22%
Epoch 287/300, Validation Loss: 1.6217, Validation Accuracy: 83.85%
Epoch 288/300, Validation Loss: 1.6231, Validation Accuracy: 83.65%
Epoch 289/300, Validation Loss: 1.6187, Validation Accuracy: 84.22%
Step 8100, Train Loss: 1.5465
Epoch 290/300, Validation Loss: 1.6161, Validation Accuracy: 84.55%
Epoch 291/300, Validation Loss: 1.6183, Validation Accuracy: 84.08%
Epoch 292/300, Validation Loss: 1.6172, Validation Accuracy: 84.38%
Step 8200, Train Loss: 1.5593
Epoch 293/300, Validation Loss: 1.6185, Validation Accuracy: 84.22%
Epoch 294/300, Validation Loss: 1.6169, Validation Accuracy: 84.15%
Epoch 295/300, Validation Loss: 1.6161, Validation Accuracy: 84.25%
Epoch 296/300, Validation Loss: 1.6186, Validation Accuracy: 84.32%
Step 8300, Train Loss: 1.5818
Epoch 297/300, Validation Loss: 1.6181, Validation Accuracy: 84.22%
Epoch 298/300, Validation Loss: 1.6189, Validation Accuracy: 83.95%
Epoch 299/300, Validation Loss: 1.6162, Validation Accuracy: 84.45%
Epoch 300/300, Validation Loss: 1.6155, Validation Accuracy: 84.45%
```

Music Genre Classification - Project

