17 novembre 2024

Mini-Projet : Traitement du son avec Python 2024-2025

ING2 MI2

Réalise par :

MILLET Jérémy

CHOMETON Maxence

BATTAIS Nathan

SIGNOUD Nathan

BRETAGNE François

ROUDET Estelle

A l'attention par :

BAHTITI Mohamed

Sommaire

1. Choix du sujet et mise en place d'une stratégie de résolution

Idées de projet :

Détection et identification d'instruments de musique

- **Applications** :
  - Transcription musicale automatique (conversion de musique en partitions).
  - Systèmes de recommandation de musique ou d'analyse musicale.
  - Applications éducatives pour apprendre la reconnaissance des instruments.
- **Travail possible** : Créer un programme qui détecte quel instrument est joué dans un enregistrement audio.

Filtrage et réduction du bruit

- **Applications** :
  - Casques à réduction de bruit active.
  - Amélioration de la qualité des appels dans les applications de vidéoconférence.
  - Systèmes de sécurité pour détecter des sons spécifiques (bruit de verre brisé, cris d'alarme).
- **Travail possible** : Concevoir un filtre de réduction de bruit en temps réel ou un filtre qui élimine un type spécifique de bruit (par exemple, un bruit de fond constant).

Traitement du son dans la reconnaissance vocale

- **Applications** :
  - Assistants vocaux (comme Siri, Alexa, Google Assistant).
  - Systèmes de commande vocale dans les voitures ou les dispositifs médicaux.
  - Transcription automatique d'entretiens ou de cours (pour les personnes malentendantes par exemple).
- **Travail possible** : Créer un modèle de reconnaissance vocale simplifié, ou explorer des méthodes de réduction du bruit pour améliorer la précision.

Sujet retenu : Application de reconnaissance musicale similaire à Shazam

Une d'application dans le domaine musical similaire à Shazam serait un système de reconnaissance musicale.

Application : Système de Reconnaissance Musicale

Objectif :

Développer une application capable d'identifier une chanson à partir d'un court extrait audio. Cela impliquerait de capturer le son ambiant, de l'analyser, et de trouver une correspondance avec une base de données de pistes musicales.

Étapes du Projet :

1. Collecte de données :

- Assemblez une base de données de chansons, en vous assurant d'avoir les fichiers audios numériques et les métadonnées associées (titre, artiste, album).

2. Extraction des caractéristiques audio :

- Utilisez des bibliothèques comme Librosa pour extraire des caractéristiques audios des fichiers, telles que :
  - Spectrogrammes
  - Fingerprints audio (empreintes digitales audio)
  - Descripteurs de fréquence

3. Création de la base de données de référence :

- Stockez les caractéristiques extraites dans une base de données optimisée pour la recherche rapide et efficace.

4. Capture et traitement de l'audio en temps réel :

- Implémentez une fonctionnalité pour capturer l'audio ambiant via le microphone.
- Traitez cette capture pour extraire ses caractéristiques.

5. Algorithme de correspondance :

- Confrontez les caractéristiques de l'extrait capturé avec celles de la base de données en utilisant des techniques de matching, telles que l'algorithme de recherche par nearest neighbors.

6. Interface utilisateur :

- Développez une interface où l'utilisateur peut activer la reconnaissance et voir les résultats (c'est-à-dire, le titre de la chanson, l'artiste, etc.).

7. Validation et optimisation :

- Testez l'application avec différents extraits dans divers environnements sonores.
- Optimisez le système pour améliorer la précision et réduire le temps de réponse.

Technologies et outils :

- Python avec des bibliothèques comme Librosa, NumPy, SciPy pour le traitement audio.
- Interface : Utiliser des outils pour construire l'interface utilisateur comme Tkinter en Python pour les applications de bureau.

2. Choix de la base de données :

GTZAN Genre Collection

Description : Ce dataset contient 1 000 morceaux de musique répartis en 10 genres différents (par exemple : rock, jazz, classique, etc.). Bien qu'il soit relativement petit, il est utilisé fréquemment pour des projets de reconnaissance musicale et d'apprentissage automatique.

Compatibilité avec Python : On peut utiliser des bibliothèques comme librosa ou pyAudioAnalysis pour extraire des caractéristiques audios de ce dataset.

C'est une bonne option pour un premier projet de reconnaissance de musique avec des genres prédéfinis.

Librosa et PyAudioAnalysis

Description : Ces deux bibliothèques Python permettent de travailler directement avec des fichiers audios, d'extraire des caractéristiques musicales et d'effectuer une analyse audio. Tu peux les utiliser avec n'importe quel jeu de données.

Compatibilité avec Python : librosa et pyAudioAnalysis sont entièrement compatibles avec Python et offrent une multitude de fonctionnalités pour extraire des descripteurs audios et des caractéristiques de signal nécessaires pour la reconnaissance musicale.

Extraction de caractéristiques audio : Utilise librosa pour extraire des caractéristiques telles que :

Spectrogrammes

MFCC

Chroma features (utile pour la reconnaissance de tonalité)

Zero crossing rate, etc.

Méthodes de comparaison : Utilise des techniques comme la recherche de similarité avec des descripteurs musicaux ou des approches basées sur des modèles d'apprentissage automatique comme les réseaux neuronaux ou les machines à vecteurs de support (SVM) pour comparer l'extrait de musique avec les chansons référencées.

Gestion des fichiers audio : Pour gérer des fichiers audios localement, on les a téléchargés depuis des plateformes comme Freesound ou GTZAN, puis on les a analysés avec librosa ou pyAudioAnalysis.

3. Bibliothèque retenue et implémentation de notre BDD et nos titres musicaux.

Pour réaliser un projet de reconnaissance musicale en Python, nous avons eu besoin de plusieurs bibliothèques pour manipuler les données audios, extraire des caractéristiques, effectuer des comparaisons et gérer les métadonnées. Voici une liste des bibliothèques essentielles que nous avons utilisées pour notre projet, avec des explications sur leur rôle :

1. Librosa

Utilisation : Librosa est une bibliothèque fondamentale pour l'analyse audio. Elle permet d'extraire des caractéristiques de signaux audio comme le spectrogramme, les MFCC (Mel-frequency cepstral coefficients), les chroma features, le pitch, etc.

Installation :

pip install librosa

Exemples d'utilisation :

Extraction de MFCC : librosa.feature.mfcc()

Calcul du spectrogramme : librosa.feature.melspectrogram()

2. PyDub

Utilisation : PyDub est une bibliothèque pour la manipulation de fichiers audio, notamment pour convertir des formats audio (par exemple WAV vers MP3), découper des fichiers ou changer leur fréquence d'échantillonnage.

Installation :

pip install pydub

Exemple d'utilisation :

Conversion de formats : AudioSegment.from_file("song.wav")

Exportation : audio.export("output.mp3", format="mp3")

3. NumPy

Utilisation : NumPy est essentiel pour la manipulation de matrices et de vecteurs en Python. Les caractéristiques audios extraites avec Librosa sont souvent sous forme de matrices que l'on manipule avec NumPy.

Installation :

pip install numpy

Exemple d'utilisation :

Traitement des vecteurs audio : numpy.array()

4. SciPy

Utilisation : SciPy fournit des fonctions pour les calculs scientifiques, comme le filtrage de signaux ou la conversion de Fourier. Elle est utile pour effectuer des transformations de signaux et pour le prétraitement des données.

Installation :

pip install scipy

Exemple d'utilisation :

Transformation de Fourier : scipy.fftpack.fft()

Filtrage de signal : scipy.signal.filtfilt()

5.Matplotlib et Seaborn (pour visualiser les données)

Utilisation : Ces bibliothèques permettront de visualiser des graphiques, comme des spectrogrammes, des courbes de MFCC, ou d'autres visualisations utiles pour l'analyse des données audio.

Installation :

pip install matplotlib seaborn

Exemple d'utilisation :

Visualisation d'un spectrogramme : plt.imshow(spec, cmap='inferno')

Visualisation de l'évolution du pitch : seaborn.lineplot()

4.Extraction des caractéristiques les plus utiles pour le traitement de la base de données.

Voir code en Annexe :

Ce code est écrit en Python et utilise la bibliothèque Librosa pour extraire des empreintes audios et afficher des spectrogrammes à partir de fichiers audio. Voici un résumé des principales fonctionnalités et étapes du code :

Spectrogramme et empreinte d'un audio :

Définition :

Dans un premier temps, grâce à la bibliothèque Librosa nous avons extrait l'empreinte de l'audio, puis on a pu récupérer le spectrogramme de ce même audio. Le spectrogramme représente visuellement l'audio en le découpant en plusieurs plages temporelles. La distribution des fréquences est réalisée en appliquant la transformée de Fourier. L'empreinte d'un audio est une représentation ponctuelle du signal ce qui permet de gagner en rapidité lors du calcul mais aussi en place de stockage. L'extraction de ces caractéristiques permet de faciliter l'analyse d'un audio.

Utilité :

Le spectrogramme va d'abord permettre à l'application de classifier et différencier les genres musicaux de la base données. L'empreinte audio va ensuite permettre d'identifier rapidement et efficacement de quelle musique il s'agit.

Pour la comparaison de deux musiques, nous comparons leurs empreintes audio via le calcul de la distance entre les cosinus des empreintes. Ainsi plus la distance tend vers 0, plus les musiques se ressemblent.

Utilisation dans l'application :

Par la suite, nous comparons une musique entrée dans l'application avec toutes les musiques de la base de données. Si la musique n'est pas dans la base de données, il n'y aura pas de correspondance. Si effectivement la musique se trouve dans notre base de données, l'application va faire correspondre la musique en entrée avec une musique de la base de données en donnant la distance entre les deux musiques. On considère pour cette étude que si la distance entre deux empreintes est inférieure à la marge d'erreur choisie au départ, alors les empreintes sont similaires.

1. Extraction de l'empreinte audio : cf Annexe 1

- La fonction `extract_empreinte` prend un fichier audio comme entrée et utilise la transformée de Fourier à courte durée (STFT) pour analyser le signal.

- Elle charge le fichier audio et le convertit en un signal numérique, puis applique la STFT pour obtenir un spectrogramme représentant les amplitudes des fréquences au fil du temps.

- L'empreinte audio est ensuite calculée en prenant la moyenne des amplitudes pour chaque fréquence, produisant ainsi un vecteur qui résume l'information audio.

2. Exemples d'utilisation :

- Le code extrait d'abord l'empreinte audio d'un exemple de trompette fourni par Librosa.

- Ensuite, on extrait les empreintes audios de deux fichiers de musique de 'The Weeknd', nommés "The Weeknd.mp3" et "The Weeknd2.mp3". Les empreintes résultantes sont affichées dans la console.

3. Affichage du spectrogramme : cf Annexe 2

- La fonction `afficher_spectrogramme` charge également un fichier audio et utilise la STFT pour créer un spectrogramme.

- Les amplitudes sont converties en décibels pour une meilleure visualisation, et le spectrogramme est affiché à l'aide de Matplotlib, permettant d'analyser visuellement l'évolution des fréquences dans le temps.

Ce code est précieux pour l'analyse et la comparaison des empreintes sonores d'enregistrements audio, utile dans des applications telles que la reconnaissance de musique ou l'analyse de la qualité du son. Les spectrogrammes offrent également une visualisation intuitive des caractéristiques sonores du contenu audio.

5. Comparaison des pistes audios et exemples

Voir code en Annexe :

Ce code en Python utilise la bibliothèque Librosa pour comparer des empreintes audio et rechercher des correspondances dans une base de données. Voici un résumé de ses principales fonctionnalités et opérations :

1. Comparaison des empreintes audio : Annexe 3/4

- La fonction `comparer_empreintesOneVSOne` compare deux empreintes audios et renvoie `True` si elles sont similaires (c'est-à-dire si la distance cosinus entre elles est inférieure à une marge d'erreur définie). Cela implique :

  - L'ajustement de la longueur des deux empreintes pour s'assurer qu'elles sont de taille identique.

  - Le calcul de la distance cosinus, qui mesure la similarité entre deux vecteurs (ici, les empreintes audio).

2. Recherche dans une base de données :

- La fonction `comparer_empreintesData` cherche à comparer une empreinte audio donnée contre tous les fichiers audio (.wav et .mp3) d'un dossier spécifié et de ses sous-dossiers. Les étapes incluent :

  - Parcourir tous les fichiers audios du dossier choisi.

  - Extraire l'empreinte de chaque fichier audio.

  - Comparer l'empreinte choisie avec chaque empreinte trouvée dans la base de données en utilisant la distance cosinus.

  - Imprimer le nom de tout fichier avec lequel une correspondance a été trouvée et retourner son chemin complet.

3. Visualisation des spectrogrammes :

- Des exemples d'utilisation de la fonction `afficher_spectrogramme` sont fournis pour visualiser les spectrogrammes de plusieurs fichiers audios, notamment "The

Weeknd.mp3", "The Weeknd2.mp3", et un exemple de trompette. Cela aide à observer comment les différentes fréquences évoluent dans le temps.

4. Cas d'utilisation :

- Le code montre comment extraire l'empreinte d'un fichier audio et rechercher des musiques similaires dans une base de données de genres musicaux, notamment en cherchant des correspondances pour des fichiers spécifiques de "The Weeknd" et d'autres genres comme le blues.

En résumé, notre code permet de réaliser une analyse comparative des empreintes audio, facilitant des applications comme la reconnaissance musicale et l'identification de fichiers audio similaires dans une vaste collection de données. Les visualisations des spectrogrammes ajoutent une dimension informative pour comprendre les caractéristiques des sons observés.

6. Interface utilisateur

Voir code en Annexe : Annexe 5/6

Notre interface utilisateur de l'application de reconnaissance musicale a été conçue pour offrir une expérience utilisateur simple et intuitive. Développée à l'aide de la bibliothèque Tkinter en Python, elle présente une approche minimaliste, privilégiant la clarté et la facilité d'utilisation.

L'IU se compose d'une fenêtre principale contenant les éléments suivants :

- Titre : Un titre clair et concis, tel que "Reconnaissance Musicale" ou "Identifiez vos Chansons !", est affiché en haut de la fenêtre pour informer immédiatement l'utilisateur de la fonctionnalité de l'application. Une police de caractères lisible et une taille appropriée améliorent l'expérience visuelle.

- Bouton "Charger une chanson" : Ce bouton principal est clairement identifié et visuellement mis en évidence (par exemple, à l'aide de couleurs contrastées). Son action est liée à la fonction `load_file_and_compare()`. Lorsqu'il est cliqué, une boîte de dialogue standard de sélection de fichier s'ouvre, permettant à l'utilisateur de choisir un fichier audio (MP3 ou WAV). Le chemin du fichier sélectionné est ensuite transmis à la fonction de traitement audio pour l'analyse.

- Bouton "Quitter" : Ce bouton, positionné de manière logique dans l'IU, permet à l'utilisateur de fermer l'application proprement. Son action est directement liée à la fonction `root.quit()` de Tkinter.

- Gestion des erreurs et du feedback utilisateur : Des messages clairs sont affichés via des boîtes de dialogue (`messagebox.showinfo()`) pour communiquer les résultats de la comparaison. En cas d'échec (fichier non trouvé, format incorrect, aucune correspondance trouvée), des messages d'erreur appropriés et concis sont fournis pour guider l'utilisateur. L'ajout d'une barre de progression pourrait améliorer l'expérience utilisateur en indiquant l'avancement du processus de comparaison.
- Design et ergonomie : L'IU adopte une approche minimaliste et épurée afin d'éviter toute surcharge d'informations visuelles. Une palette de couleurs cohérente et un espacement approprié entre les éléments améliorent la lisibilité et la convivialité.

L'architecture de l'interface, basée sur un simple cadre (`tk.Frame`) et des éléments de base de Tkinter, privilégie la simplicité et la maintenabilité du code. L'IU est conçue pour être facilement adaptable et extensible, permettant l'intégration de fonctionnalités supplémentaires, comme l'affichage des résultats dans une liste ou la gestion d'une base de données plus importante.

7.Tests et Exemples

1. Visualisation des Spectrogrammes :

Nous réalisons une analyse visuelle en affichant les spectrogrammes de différents fichiers audios à l'aide de la fonction `afficher_spectrogramme`. Cela permet d'observer les composants fréquentiels au fil du temps pour chaque morceau audio.

Exemples :

- The Weeknd.mp3 : Un spectrogramme pour ce fichier est généré pour visualiser ses motifs fréquenciels uniques.

- The Weeknd2.mp3 : De même, le spectrogramme d'un deuxième fichier musical est visualisé.

- Exemple de `librosa` (`trumpet`) : En visualisant un échantillon connu, nous pouvons le comparer avec nos fichiers musicaux à titre éducatif.

2. Extraction et Comparaison d'Empreintes Audio :

Deux fonctions principales : `extract_empreinte` et `comparer_empreintesOneVSOne` sont testées.

Extraction et Comparaison Directe :

- Les empreintes sont extraites pour deux versions audio (`The Weeknd.mp3` et `The Weeknd2.mp3`) et comparées. Étant donné leurs similarités, notre fonction de comparaison devrait retourner `True`, indiquant une correspondance étroite.

- Une autre comparaison entre `The Weeknd.mp3` et un exemple de trompette est réalisée pour démontrer la capacité de notre fonction à différencier les sons non liés, en s'attendant à un résultat `False`.

3. Comparaison avec une Base de Données :

La fonction `comparer_empreintesData` permet des tests plus approfondis en comparant une empreinte audio avec une base de données de fichiers audio stockés :

- Exemple 1 : Un fichier audio spécifique (`blues.00016.wav`) est analysé par rapport à la base de données. Si un fichier similaire est trouvé, il est identifié, sinon, il est confirmé qu'il n'y a pas de correspondance proche.

- Exemple 2 : Pour `The Weeknd.mp3`, nous démontrons la polyvalence de notre système à traiter un audio connu et à évaluer la similitude à travers différents genres et styles présents dans notre répertoire de la base de données.

Ces exemples servent à valider l'implémentation des fonctionnalités de traitement audio fournies par le code. Ils démontrent non seulement l'efficacité des algorithmes à extraire des caractéristiques significatives des audios, mais aussi leur utilité dans des applications concrètes, comme les systèmes de récupération audio. En réalisant ces tests, nous assurons la fiabilité et la précision pour les futurs utilisateurs ou clients qui pourraient utiliser ce système pour l'empreinte et la comparaison audio.

8. Annexes

Annexe 1 : extraction de l'empreinte du son

```python
#extraction de l'empreinte du son, renvoie l'empreinte
def extract_empreinte(audio, taille_fenetre=2048, pas_saut=512):
    # Charge l'audio spécifié et le convertit en un signal numérique
    signal, hz = lb.load(audio)

    # applique fournier sur le signal,
    #cad : recupere le spectogramme du signal
    stft = np.abs(lb.stft(signal, n_fft=taille_fenetre, hop_length=pas_saut))

    # recupere moyenne des amplitudes de chaque fréquence pour simplifier chaque ligne
    fingerprint = np.mean(stft, axis=1)

    return fingerprint

#audio utilisé pour voir si ça renvoie bien faux pour la comparaisonOneVSOne
audio_exemple = lb.example('trumpet')
fingerprint  = extract_empreinte(audio_exemple)
print("Empreinte audio trompette :")
print(fingerprint)

#audio weeknd1
musique = "The Weeknd.mp3"
print("Empreinte audio de theweek end :")
extracttheweekd1= extract_empreinte(musique)
print(extracttheweekd1)

#audio weeknd2
musique2 = "The Weeknd2.mp3"
print("Empreinte audio de theweek end 2 :")
extracttheweekd2=extract_empreinte(musique2)
print(extracttheweekd2)
```

Annexe 2 : affichage du spectrogramme

```python
#-------------affichage du spectre avec matplotlib-----------------------------------------------------------------------
def afficher_spectrogramme(audio, taille_fenetre=2048, pas_saut=512):
    # Charge l'audio
    signal, sr = lb.load(audio)

    # Applique la STFT pour obtenir le spectrogramme
    D = np.abs(lb.stft(signal, n_fft=taille_fenetre, hop_length=pas_saut))

    # Convertir l'amplitude en décibels pour une meilleure visualisation
    DB = lb.amplitude_to_db(D, ref=np.max)

    # Afficher le spectrogramme avec matplotlib
    plt.figure(figsize=(12, 8))
    plt.imshow(DB, aspect='auto', cmap='inferno', origin='lower',
               extent=[0, DB.shape[-1], 0, DB.shape[0]])
    plt.colorbar(format="%+2.0f dB")
    plt.title(f"Spectrogramme de {audio}")
    plt.xlabel('Temps')
    plt.ylabel('Fréquence')
    plt.show()
```

Annexe 3 : Comparaison des empreintes audio

```python
#---------------compare deux empreintes de son et renvoie vrai s'ils sont identiques----------------------------------------
def comparer_empreintesOneVSOne(empreinte1, empreinte2, margeerreur=0.01):
    # Ajuster la taille des empreintes pour qu'elles soient identiques
    min_len = min(len(empreinte1), len(empreinte2))
    empreinte1, empreinte2 = empreinte1[:min_len], empreinte2[:min_len]

    # Calculer la distance du cosinus entre les deux vecteurs (plus elle est proche de 0, plus elles sont similaires)
    distance = cosine(empreinte1, empreinte2)

    # Retourne si la distance est inférieure à la tolérance définie
    if (distance<margeerreur):
        return True

    else :
        return False
```

Annexe 4 : Comparaison d'une musique dans la BDD

```python
#-------------compare une musique d'entrée avec tout les fichiers .mp3 et .wav des sous dossiers du dossier choisi--------------
def comparer_empreintesData(empreinte_choisie, dossier_base, marge_erreur=0.01):

    # Parcourt tous les sous-dossiers et fichiers dans le dossier principal
    for racine, _, fichiers in os.walk(dossier_base):
        for fichier in fichiers:
            # Construit le chemin complet du fichier audio
            chemin_fichier = os.path.join(racine, fichier)

            # Vérifie que le fichier est bien un fichier audio (ici, on filtre pour .wav et .mp3)
            if fichier.endswith(('.wav', '.mp3')):
                # Extraction de l'empreinte du fichier dans la base de données
                empreinte_fichier = extract_empreinte(chemin_fichier)

                # Ajuste les longueurs des empreintes pour qu'elles soient identiques
                min_len = min(len(empreinte_choisie), len(empreinte_fichier))
                empreinte_choisie, empreinte_fichier = empreinte_choisie[:min_len], empreinte_fichier[:min_len]

                # Calcule la distance de cosinus entre les deux empreintes
                distance = cosine(empreinte_choisie, empreinte_fichier)

                # Si la distance est inférieure à la marge d'erreur, considère les empreintes comme similaires
                if distance < marge_erreur:
                    nom_fichier = os.path.basename(chemin_fichier)
                    print(f"Correspondance trouvée avec : {nom_fichier} (distance : {distance})")
                    return chemin_fichier  # Retourne le chemin du fichier correspondant

    print("Aucune correspondance trouvée dans le dossier.")
    return None  # Aucun fichier similaire trouvé
```

Annexe 5 : Interface utilisateur

```
import tkinter as tk
from tkinter import filedialog, messagebox
import librosa as lb
import numpy as np
from scipy.spatial.distance import cosine
import os

def extract_empreinte(audio, taille_fenetre=2048, pas_saut=512):
    signal, _ = lb.load(audio)
    stft = np.abs(lb.stft(signal, n_fft=taille_fenetre, hop_length=pas_saut))
    fingerprint = np.mean(stft, axis=1)
    return fingerprint

def comparer_empreintesOneVSOne(empreinte1, empreinte2, margeerreur=0.01):
    min_len = min(len(empreinte1), len(empreinte2))
    empreinte1, empreinte2 = empreinte1[:min_len], empreinte2[:min_len]
    distance = cosine(empreinte1, empreinte2)
    return distance < margeerreur

def comparer_empreintesData(empreinte_choisie, dossier_base, marge_erreur=0.01):
    for racine, _, fichiers in os.walk(dossier_base):
        for fichier in fichiers:
            if fichier.endswith(('.wav', '.mp3')):
                chemin_fichier = os.path.join(racine, fichier)
                empreinte_fichier = extract_empreinte(chemin_fichier)
                min_len = min(len(empreinte_choisie), len(empreinte_fichier))
                empreinte_choisie, empreinte_fichier = empreinte_choisie[:min_len],
empreinte_fichier[:min_len]
                distance = cosine(empreinte_choisie, empreinte_fichier)
                if distance < marge_erreur:
                    return chemin_fichier
    return None

def load_file_and_compare():
    file_path = filedialog.askopenfilename(filetypes=[("Audio Files", "*.mp3;*.wav")])
    if file_path:
        empreinte = extract_empreinte(file_path)
        dossier_base = "chemin_vers_votre_dossier"  # Remplissez le chemin vers votre dossier de
musique
        fichier_similaire = comparer_empreintesData(empreinte, dossier_base)
        if fichier_similaire:
            nom_fichier = os.path.basename(fichier_similaire)
            messagebox.showinfo("Résultat", f"Musique similaire trouvée : {nom_fichier}")
        else:
            messagebox.showinfo("Résultat", "Aucune musique similaire trouvée dans la base de
données.")
```

Annexe 6 : Affichage

```
# Créer la fenêtre principale
root = tk.Tk()
root.title("Reconnaissance Audio")
root.geometry("400x400")
root.configure(bg="#f0f0f0")  # Couleur de fond agréable

# Créer un cadre pour l'organisation
frame = tk.Frame(root, bg="white", bd=2, relief=tk.GROOVE)
frame.place(relx=0.5, rely=0.5, anchor='center', width=350, height=300)

# Ajouter un titre
title_label = tk.Label(frame, text="Reconnaître une chanson", font=("Helvetica", 18), bg="white",
fg="#333")
title_label.pack(pady=10)

# Créer un bouton pour charger un fichier audio
load_button = tk.Button(frame, text="Charger une chanson", command=load_file_and_compare,
                        font=("Helvetica", 14), bg="#1DB954", fg="white", padx=20, pady=10)
load_button.pack(pady=20)

# Créer un bouton pour quitter l'application
quit_button = tk.Button(frame, text="Quitter", command=root.quit,
                        font=("Helvetica", 14), bg="#FF4500", fg="white", padx=20, pady=10)
quit_button.pack(pady=10)

# Lancer l'interface
root.mainloop()
```

Annexe 7 : Exemples et tests

```
#-------------------------------------Exemples & Tests ---------------------------------------------------------------
# Exemple utilisation affichage spectogramme :
musique = "The Weeknd.mp3"
afficher_spectrogramme(musique)
musique2 = "The Weeknd2.mp3"
afficher_spectrogramme(musique2)
musique3 = lb.example('trumpet')
afficher_spectrogramme(musique3)

#exemple utilisation compare_empreinte :
#compare deux sons similaires
print(comparer_empreintesOneVSOne(extracttheweekd1,extracttheweekd2))
#comparaison de the week end avec trumpet
print(comparer_empreintesOneVSOne(extracttheweekd1,fingerprint))


#exemple1 BDD
# Empreinte de la musique d'entrée
empreinte_entree = extract_empreinte(r"H:\Documents\ING2 CYTECH\ING2-TraitementSignal\Projet Traitement Signal\Data\genres_original\blues\blues.00016.wav")
# Chemin du dossier contenant les empreintes de la base de données
dossier_base = r"H:\Documents\ING2 CYTECH\ING2-TraitementSignal\Projet Traitement Signal\Data\genres_original"
# Appel de la fonction
fichier_similaire = comparer_empreintesData(empreinte_entree, dossier_base)
nom_fichier = os.path.basename(fichier_similaire)
if fichier_similaire:
    print(f"Musique similaire trouvée : {nom_fichier}")
else:
    print("Aucune musique similaire trouvée dans la base de données.")
```