

---

# Rapport du projet nouvelles architectures

*De*

**Troisième année en Génie Informatique**

*Présenté et publié le 13/12/2023*

*Par*

**Chakroun Asma**

---

## **Conception d'un environnement ML de classification musicale avec une intégration continue**

---

*Proposé par*

**Mr Mehrez Boulaares**

**Année universitaire : 2023-2024**

---

# Table de Matière

Chapitre 1	Présentation Globale.....	4
1.	Objectif.....	4
2.	Description de la Dataset.....	4
3.	Les parties su projets : .....	5
Chapitre 2	Architecture du projet.....	6
	Introduction : .....	6
1.	L'architecture du projet : .....	6
2.	Les composants du projet : .....	7
Chapitre 3	Contenu.....	9
	Introduction : .....	9
1.	Frontend_service : .....	9
2.	Svm_Service : .....	11
	.....	12
4.	Docker-Compose : .....	14
Chapitre 4	Réalisation .....	15
	Introduction : .....	15
1.	Précision de l'algorithme SVM : .....	15
2.	Précision de l'algorithme VGG19 : .....	15
3.	Spectrogramme de l'algorithme VGG19 : .....	16
4.	Interface de résultat .....	16
Conclusion :	.....	18

## Table de figure

Figure 1 Contenu DataSet .....	5
Figure 2 Architecture du projet .....	7
Figure 3 Le fichier Flask .....	9
Figure 4 Le fichier __init__.py .....	10
Figure 5 fichier .html.....	10
Figure 6 Dockerfile de la partie frontend .....	11
Figure 7 fichier requirement pour la partie frontend .....	11
Figure 8 SVM algorithme .....	11
Figure 9 DockerFile svm.....	12
Figure 10 Requirement.txt svm.....	12
Figure 11 VGG19 Algorithme .....	12
Figure 12 Spectrogramme VGG.....	13
Figure 13 Dockerfile VGG.....	13
Figure 14 requirement VGG.....	13
Figure 15 Docker-compose.yml.....	14
Figure 16 Précision algorithme SVM.....	15
Figure 17 Précision VGG19 .....	15
Figure 18 Spectrogramme VGG19 .....	16
Figure 19 Interface de résultat.....	16
Figure 20 Résultat de prédiction .....	17
Figure 21 Choix audio.....	17

# Chapitre 1      Présentation Globale

## 1. Objectif

L'objectif de ce projet est de créer un environnement de Machine Learning permettant la classification des genres musicaux à partir de fichiers audios au format WAV, en utilisant Python, Flask pour les services web, Les modèles SVM et VGG19 pour la partie Machine Learning, et Docker pour la conteneurisation.

## 2. Description de la Dataset

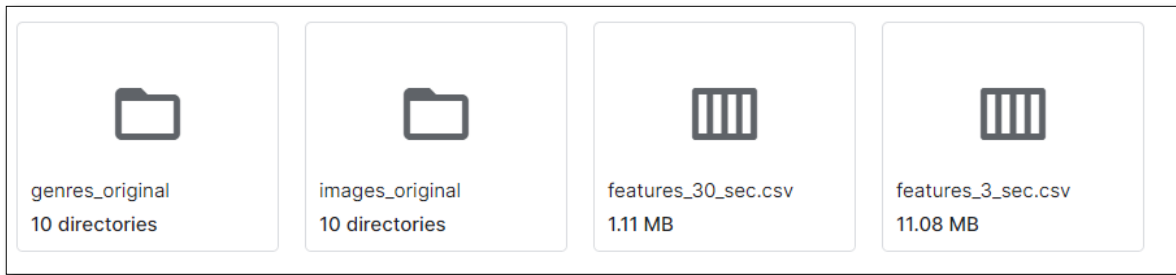
Cette DataSet offre une diversité de genres musicaux avec une collection des fichiers audios .WAV des musiques pour l'entraînement du model, et deux fichiers CSV fournissant des caractéristiques audios pour chaque chanson, permettant ainsi d'exploiter ces données dans des modèles de classification.

Le contenu de notre DataSet est :

➤ **2 dossiers :**

- Genres originaux : Une collection de 10 genres comprenant chacun 100 fichiers audio, tous d'une durée de 30 secondes.
- Images originales : Une représentation visuelle pour chaque fichier audio. Une façon de classifier les données est à travers les réseaux neuronaux.

- **2 fichiers CSV :** Contenant des caractéristiques des fichiers audio. Un fichier a, pour chaque chanson (de 30 secondes), une moyenne et une variance calculées sur plusieurs caractéristiques qui peuvent être extraites d'un fichier audio. L'autre fichier a la même structure, mais les chansons ont été préalablement divisées en fichiers audio de 3 secondes (augmentant ainsi 10 fois la quantité de données alimentées dans nos modèles de classification).



*Figure 1 Contenu DataSet*

### 3. Les parties su projets :

Notre projet est reparti sur plusieurs parties pertinentes :

- **Intégration de la technologie Docker :**

Les conteneurs Docker simplifient la création d'environnements de Machine Learning isolés et portables. Ils facilitent également le déploiement et la gestion des services de classification en encapsulant toutes les dépendances nécessaires, assurant ainsi une portabilité maximale et une cohérence entre les environnements de développement et de production.

- **Services web Flask :**

La mise en place des services distincts SVM\_service et VGG19\_service avec Flask simplifie la classification des genres musicaux. Ces services, conçus pour recevoir des fichiers audios au format base64, offrent une solution efficace et modulaire, intégrable facilement avec des conteneurs Docker.

- **Classification**

L'utilisation des modèles de Machine Learning, tels que SVM et VGG19, au sein des services SVM\_service et vgg19\_service, permet la classification précise des genres musicaux à partir des fichiers audios fournis. Ces services offrent une interface permettant de soumettre des fichiers audios, traitant ensuite ces données pour extraire des caractéristiques pertinentes. Ces caractéristiques servent ensuite d'entrées pour les modèles de classification. L'approche adoptée permet une classification multi-classes des pistes audios en fonction des genres musicaux préalablement définis.

- **Orchestration avec Docker-compose :**

Création d'un fichier Docker-compose pour orchestrer et gérer les services Flask, les modèles de classification et autres composants nécessaires.

- **Environnement d'intégration continue et de test avec Jenkins :**

Configuration d'un environnement d'intégration continue avec Jenkins pour automatiser les tests, vérifier les mises à jour du code et évaluer les performances de l'application.

## **Chapitre 2      Architecture du projet**

### **Introduction :**

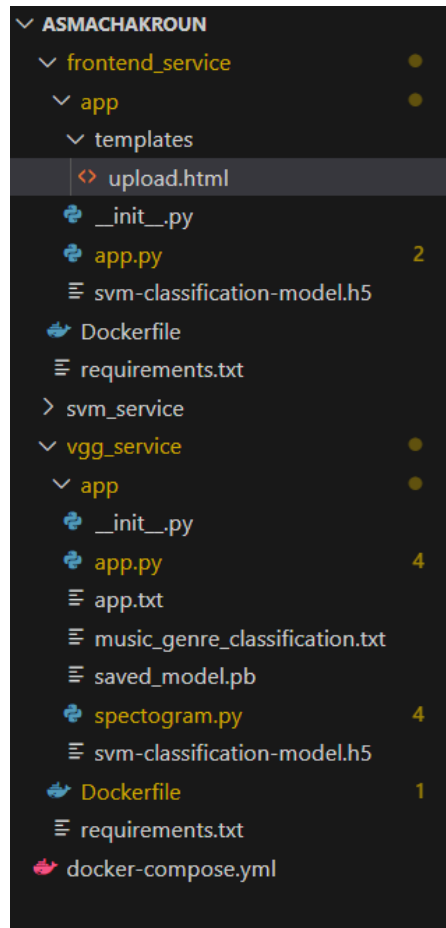
L'infrastructure de l'application s'appuie sur l'utilisation des conteneurs Docker afin de fournir une plateforme d'hébergement pour deux services Flask distincts, SVM\_service et vgg19\_service. Ces services interagissent étroitement avec les modèles de classification (SVM et VGG19) pour réaliser la classification des genres musicaux à partir de fichiers audios. La coordination et la gestion de ces services dans un environnement homogène sont assurées par l'utilisation du fichier Docker-compose.yml .

### **1. L'architecture du projet :**

Dans notre projet, on a créé 3 dossiers principaux :

- Svm\_service
- Vgg\_service
- Frontend\_service

Dans la figure ci-dessous vous trouvez les fichiers réalisés dans notre projet.



*Figure 2 Architecture du projet*

## 2. Les composants du projet :

### 2.1 Implémentation des services Flask :

- Implémentation détaillée des services SVM et VGG19 :

#### **SVM\_service :**

Ce service accepte des fichiers audios au format base64, les traite pour extraire des caractéristiques significatives, puis utilise le modèle SVM pour classifier le genre musical.

#### **Vgg19\_service :**

Similaire au SVM\_service, ce service prend en charge des fichiers audio au format base64, extrait des caractéristiques spécifiques, et utilise le modèle VGG19 (un réseau neuronal convolutif) pour la classification.

- Interaction avec les modèles de classification et les fichiers audio :

### **Services Flask :**

Ils opèrent comme des interfaces, recevant les fichiers audios et transmettant ces données aux modèles appropriés.

### **Modèles de classification :**

Les services interagissent avec des modèles préalablement entraînés pour obtenir des prédictions de classification à partir des caractéristiques extraites des fichiers audios.

## **2.2 Création du Docker-compose :**

- Création image docker :

### **Création du Dockerfile :**

Le Dockerfile définit les étapes nécessaires à la construction de l'image Docker. Il inclut l'installation des dépendances, la configuration des services Flask, ainsi que l'ajout des modèles de classification.

### **Construction de l'image :**

Le processus de création de l'image s'appuie sur l'utilisation des commandes Docker build. Ces commandes sont exécutées en se basant sur les instructions définies dans le Dockerfile, permettant ainsi la construction efficace de l'image Docker.

- Principes pour garantir la portabilité et la faisabilité :

### **Dépendances explicites :**

Toutes les dépendances requises sont clairement déclarées dans le Dockerfile, garantissant ainsi la reproductibilité cohérente de l'environnement.

### **Recours aux volumes :**

L'utilisation de volumes Docker permet de rendre les données persistantes, assurant ainsi la portabilité de l'environnement entre diverses machines hôtes.



## Chapitre 3      Contenu

### Introduction :

Dans cette section, nous examinerons de manière approfondie chaque service et son mode opératoire.

#### 1. Frontend\_service :

Les figures ci-dessous contiennent les composants utilisés dans le service frontend.

La figure suivante présente le service Flask utilisé.

```
1  from flask import Flask, render_template, request
2  import requests
3
4  app = Flask(__name__)
5
6  svm_service_url = 'http://svm_service:6000'
7
8  @app.route('/')
9  def hello_world():
10     return render_template('upload.html')
11
12  @app.route('/upload', methods=['POST'])
13  def classify():
14     if 'musicFile' not in request.files:
15         return "No file provided"
16
17
18     # Get the uploaded file
19     music_file = request.files['musicFile']
20
21     # Save the file to the shared volume
22     file_path = '/Nouvarch/shared_volume/' + music_file.filename
23     music_file.save(file_path)
24
25     # Send the file to svm_service
26     files = {'musicFile': (music_file.filename, open(file_path, 'rb'))}
27
28     response = requests.post(f'{svm_service_url}/classify', files=files)
29
30     # Assuming svm_service returns a JSON response
31     response_data = response.json()
32
33
```

*Figure 3 Le fichier Flask*

La figure ci-dessous présente le fichier `__init__.py` qui sert à initialiser notre application Flask, et le fait de l'exécuter directement lance le serveur de développement Flask.

```

frontend_service > app > _init_.py > ...
1  # app/_init_.py
2
3  from flask import Flask
4
5  # Create a Flask application instance
6  app = Flask(__name__)
7
8  # Import the main application module
9  from app import app
10
11
12
13 if __name__ == '__main__':
14     app.run()
15

```

*Figure 4 Le fichier \_\_init\_\_.py*

La figure suivante présente le fichier front.html qui sert à afficher le résultat.

```

1  from flask import Flask, render_template, request
2  import requests
3
4  app = Flask(__name__)
5
6  svm_service_url = 'http://svm_service:6000'
7
8  @app.route('/')
9  def hello_world():
10     return render_template('upload.html')
11
12 @app.route('/upload', methods=['POST'])
13 def classify():
14     if 'musicFile' not in request.files:
15         return "No file provided"
16
17
18     # Get the uploaded file
19     music_file = request.files['musicFile']
20
21     # Save the file to the shared volume
22     file_path = '/Nouvarch/shared_volume/' + music_file.filename
23     music_file.save(file_path)
24
25     # Send the file to svm_service
26     files = {'musicFile': (music_file.filename, open(file_path, 'rb'))}
27
28     response = requests.post(f'{svm_service_url}/classify', files=files)
29
30     # Assuming svm_service returns a JSON response
31     response_data = response.json()
32
33

```

*Figure 5 fichier .html*

Les 2 figures ci-dessous représentent le dockerFile et requirement.txt :

```
1 FROM python:3-alpine3.15
2
3 WORKDIR ./
4
5 COPY . .
6
7 ENV STATIC_URL /static
8 ENV STATIC_PATH /app/static
9
10
11 COPY ./requirements.txt /requirements.txt
12 RUN pip install -r /requirements.txt
13
14 EXPOSE 3000
15
16 CMD ["python", "/app/app.py"]
```

*Figure 6 Dockerfile de la partie frontend*

```
Flask==2.0.1
Werkzeug==2.0.1
requests==2.26.0
```

*Figure 7 fichier requirement pour la partie frontend*

## 2. Svm\_Service :

Les figures ci-dessous contiennent les composants utilisés dans le service SVM.

La figure suivante présente l'algorithme SVM utilisé.

```
svm_service > app > app.py > ...
1 # svm_service/app.py
2 from flask import Flask, request, jsonify
3 import time
4 import os
5 import librosa
6 import numpy as np
7 from tensorflow.keras.models import load_model
8
9 app = Flask(__name__)
10 # Get the absolute path to the directory of this script
11 base_path = os.path.abspath(os.path.dirname(__file__))
12
13 # Load the pre-trained model using joblib
14 model_path = os.path.join(base_path, "svm-classification-model.h5")
15 model = load_model(model_path)
16
17 genre_dict={0: 'blues', 1: 'classical', 2: 'country', 3: 'disco', 4: 'hiphop',
18
19 # Function to retrieve the latest audio file from a directory
20 def get_latest_audio_file(directory):
21     files = [os.path.join(directory, f) for f in os.listdir(
22         directory) if f.endswith('.wav')] # Change extension if different
23     if files:
24         # Get the latest file based on creation time
25         return max(files, key=os.path.getctime)
26     else:
27         return None
28
29 def extract_features(file_path):
30     # Load audio file with librosa
31     audio, sample_rate = librosa.load(file_path, res_type='kaiser_fast')
32     # Extract MFCCs and other features...
33     mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=20)
```

*Figure 8 SVM algorithme*

Les 2 figures ci-dessous représentent le dockerFile et requirement.txt :

```
FROM python:3.9-slim-buster

WORKDIR ./

ENV STATIC_URL /static
ENV STATIC_PATH /app/static

COPY ./requirements.txt /requirements.txt
RUN pip install -r /requirements.txt

COPY /app/svm-classification-model.h5 /app/svm-classification-model.h5

COPY . .

EXPOSE 6000

CMD ["python", "/app/app.py"]
```

*Figure 9 DockerFile svm*

```
svm_service > requirements.txt
1  Flask==2.0.1
2  Werkzeug==2.0.1
3  requests==2.26.0
4  librosa==0.10.1
5  numpy==1.26.2
6  tensorflow==2.15.0
```

*Figure 10 Requirement.txt svm*

### 3. Vgg\_Service :

Les figures ci-dessous contiennent les composants utilisés dans le service VGG19.

La figure suivante présente l'algorithme VGG19 utilisé.

```
vgg_service / app / app.py
1  from flask import Flask, request, jsonify
2  import tensorflow as tf
3  import numpy as np
4  import librosa
5
6  app = Flask(__name__)
7
8  # Charger le modèle VGG
9  vgg_service_url = 'http://vgg_service:7000'
10 model_path = '/app/vgg_service'
11 model = tf.saved_model.load(model_path)
12
13 classes = ["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "pop", "re
14
15 @app.route('/predict_vgg', methods=['POST'])
16 def predict_vgg():
17     try:
18         # Assurez-vous que 'audio' est la clé correcte dans la requête multipart
19         file = request.files['audio']
20         waveform, sr = librosa.load(file, sr=16000)
21
22         # ... Traitement des caractéristiques audio ...
23
24         # Créer le tenseur d'entrée
25         inp = tf.constant(np.array([waveform]), dtype='float32')
26
27         # Faire la prédiction
28         class_scores = model(inp)[0].numpy()
29         predicted_class = classes[class_scores.argmax()]
30
31         # Retourner le résultat
32         return jsonify({'prediction': predicted_class})
33     except Exception as e:
```

*Figure 11 VGG19 Algorithme*

La figure-ci-dessous représente l'algorithme qui affiche le spectrogramme des audios pour améliorer la classification :

```

vgg_service > app > spectrogram.py > ...
1  import os
2  import librosa
3  import librosa.display
4  import matplotlib.pyplot as plt
5  import numpy as np
6
7  def create_spectrogram_for_file(audio_path, output_path):
8      # Charger le fichier audio
9      y, sr = librosa.load(audio_path, sr=None)
10
11     # Créer le spectrogramme mel
12     spec = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=2048, hop_length=512)
13     spec_db = librosa.power_to_db(spec, ref=np.max)
14
15     # Tracer et sauvegarder le spectrogramme en tant qu'image PNG
16     plt.figure(figsize=(8, 6))
17     librosa.display.specshow(spec_db, sr=sr, hop_length=512, x_axis='time', y_axis='mel')
18     plt.colorbar(format='%+2.0f dB')
19     plt.title(f"Spectrogram - {os.path.basename(audio_path)}")
20     plt.savefig(output_path, format='png')
21     plt.close()
22
23  def create_spectrograms(dataset_path, output_images_dir):
24      # Parcourir tous les sous-répertoires (genres musicaux)
25      for genre in os.listdir(dataset_path):
26          genre_path = os.path.join(dataset_path, genre)
27
28          # S'assurer que l'élément est un dossier
29          if os.path.isdir(genre_path):
30              # Créer un sous-répertoire pour les images de ce genre
31              genre_output_dir = os.path.join(output_images_dir, genre)
32              os.makedirs(genre_output_dir, exist_ok=True)
33

```

*Figure 12 Spectrogramme VGG*

Les 2 figures ci-dessous représentent le dockerFile et requirement.txt :

```

vgg_service > Dockerfile > ...
1  FROM python:3.9-slim-buster
2
3  WORKDIR ./
4
5  ENV STATIC_URL /static
6  ENV STATIC_PATH /app/static
7
8  COPY ./requirements.txt /requirements.txt
9  RUN pip install -r /requirements.txt
10
11  COPY /app/saved_model.pb /app/saved_model.pb
12
13  COPY . .
14
15  EXPOSE 7000
16
17  CMD ["python", "/app/app.py"]
18

```

*Figure 13 Dockerfile VGG*

```

vgg_service > requirements.txt
1  Flask==2.0.1
2  Werkzeug==2.0.1
3  requests==2.26.0
4  librosa==0.10.1
5  numpy==1.26.2
6  tensorflow==2.15.0

```

*Figure 14 requirement VGG*

## 4. Docker-Compose :

La figure ci-dessous represente le dockercompose.yml .

```
docker-compose.yml
1  version: '3'
2  services:
3    frontend_service:
4      build:
5        context: ./frontend_service
6        dockerfile: Dockerfile
7      ports:
8        - "3000:3000"
9      volumes:
10       - shared_volume:/Nouvarch/shared_volume
11      environment:
12        - FLASK_ENV=development
13
14    svm_service:
15      build:
16        context: ./svm_service
17        dockerfile: Dockerfile
18      ports:
19        - "6000:6000"
20      volumes:
21        - shared_volume:/Nouvarch/shared_volume
22      depends_on:
23        - frontend_service
24
25    vgg_service:
26      build:
27        context: ./vgg_service
28        dockerfile: Dockerfile
29      ports:
30        - "7000:7000"
31      volumes:
32        - shared_volume:/Nouvarch/shared_volume
33      depends_on:
34        - frontend_service
35
36  volumes:
37    shared_volume:
38
```

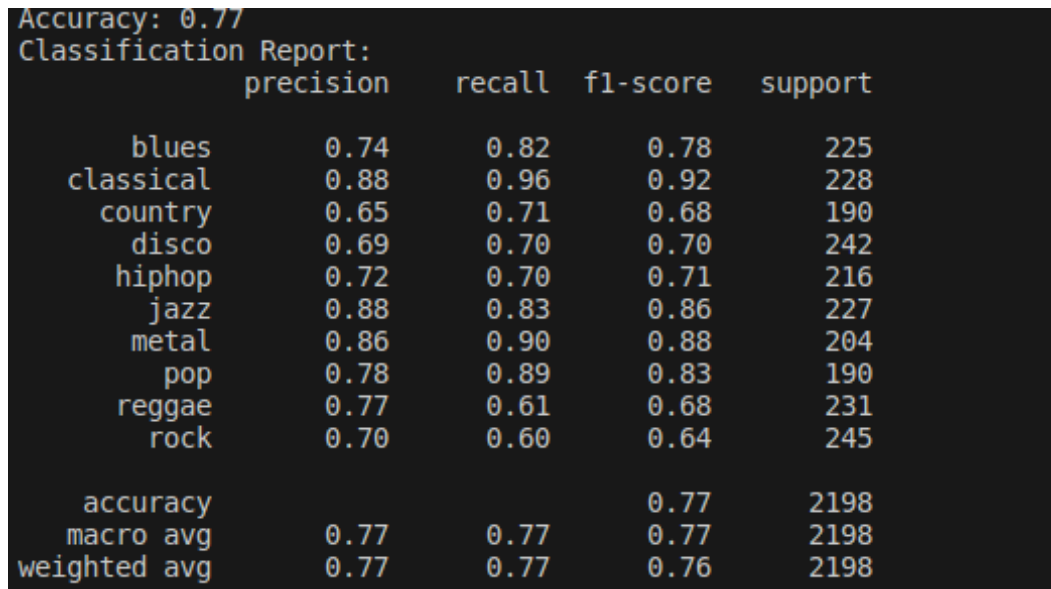
*Figure 15 Docker-compose.yml*

## Chapitre 4 Réalisation

### Introduction :

Dans cette partie on va voir les interfaces réaliser dans notre projet et les résultat affichés.

#### 1. Précision de l'algorithme SVM :



```
Accuracy: 0.77
Classification Report:
              precision    recall  f1-score   support

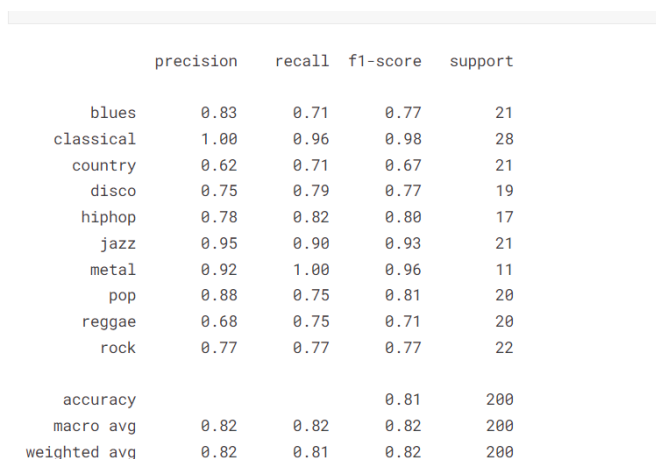
   blues         0.74      0.82      0.78        225
  classical      0.88      0.96      0.92        228
   country      0.65      0.71      0.68        190
    disco       0.69      0.70      0.70        242
   hiphop       0.72      0.70      0.71        216
    jazz        0.88      0.83      0.86        227
    metal       0.86      0.90      0.88        204
    pop         0.78      0.89      0.83        190
   reggae       0.77      0.61      0.68        231
    rock        0.70      0.60      0.64        245

 accuracy              0.77              2198
 macro avg           0.77      0.77      0.77      2198
 weighted avg        0.77      0.77      0.76      2198
```

*Figure 16 Précision algorithme SVM*

#### 2. Précision de l'algorithme VGG19 :

On présente dans la figure ci-dessous le résultat de précision de l'algorithme VGG19



```
              precision    recall  f1-score   support

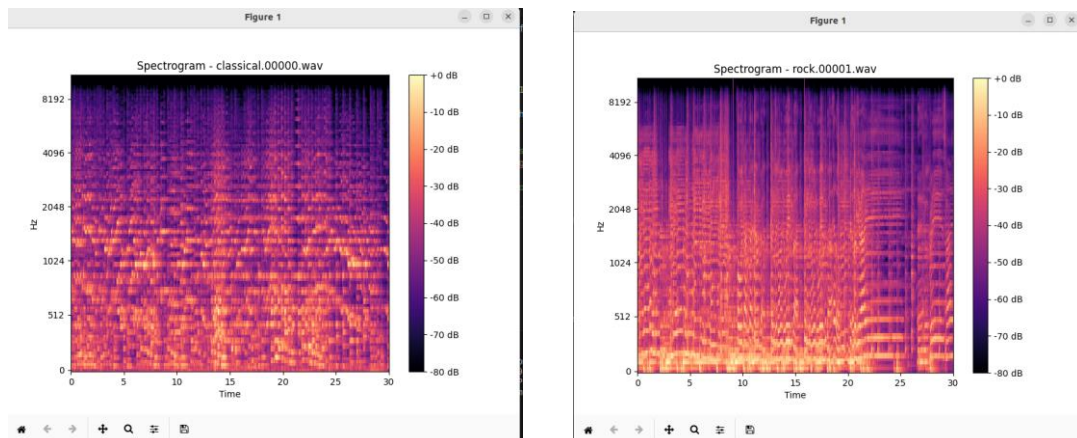
   blues         0.83      0.71      0.77        21
  classical      1.00      0.96      0.98        28
   country      0.62      0.71      0.67        21
    disco       0.75      0.79      0.77        19
   hiphop       0.78      0.82      0.80        17
    jazz        0.95      0.90      0.93        21
    metal       0.92      1.00      0.96        11
    pop         0.88      0.75      0.81        20
   reggae       0.68      0.75      0.71        20
    rock        0.77      0.77      0.77        22

 accuracy              0.81              200
 macro avg           0.82      0.82      0.82      200
 weighted avg        0.82      0.81      0.82      200
```

*Figure 17 Précision VGG19*

### 3. Spectrogramme de l'algorithme VGG19 :

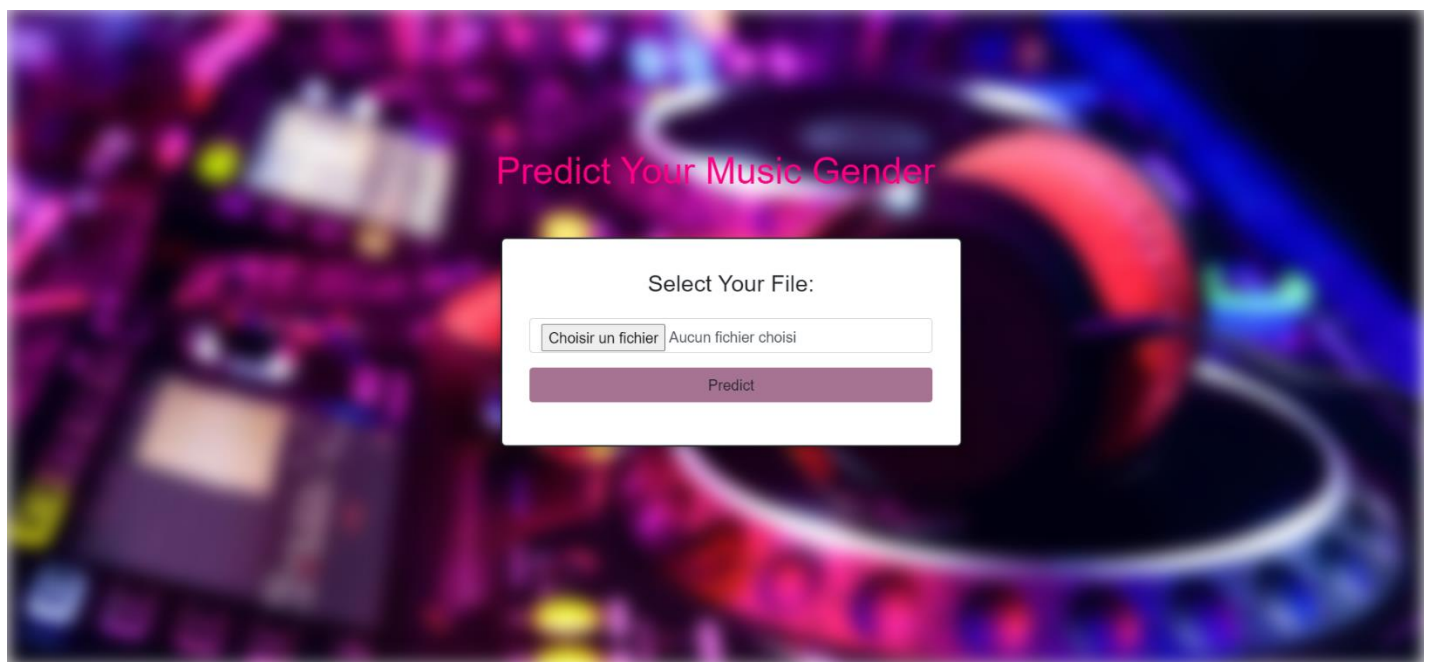
Ces 2 figures ci-dessous représente 2 exemples pour les spectrogrammes de l'algorithme vgg19.



*Figure 18 Spectrogramme VGG19*

### 4. Interface de résultat

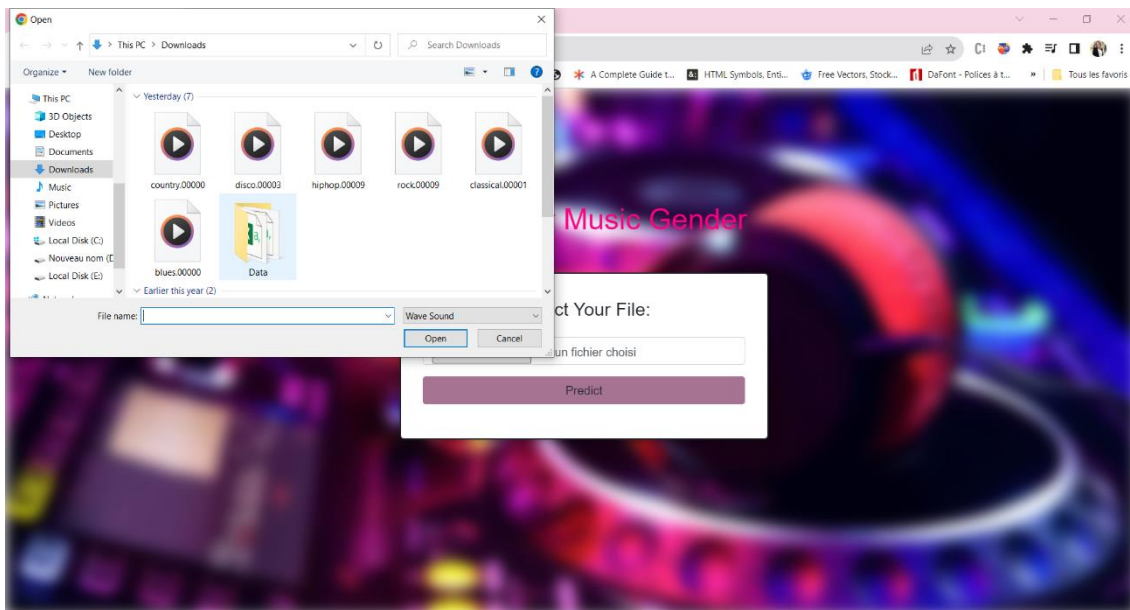
La figure ci-dessous représente l'interface de résultat.



*Figure 19 Interface de résultat*

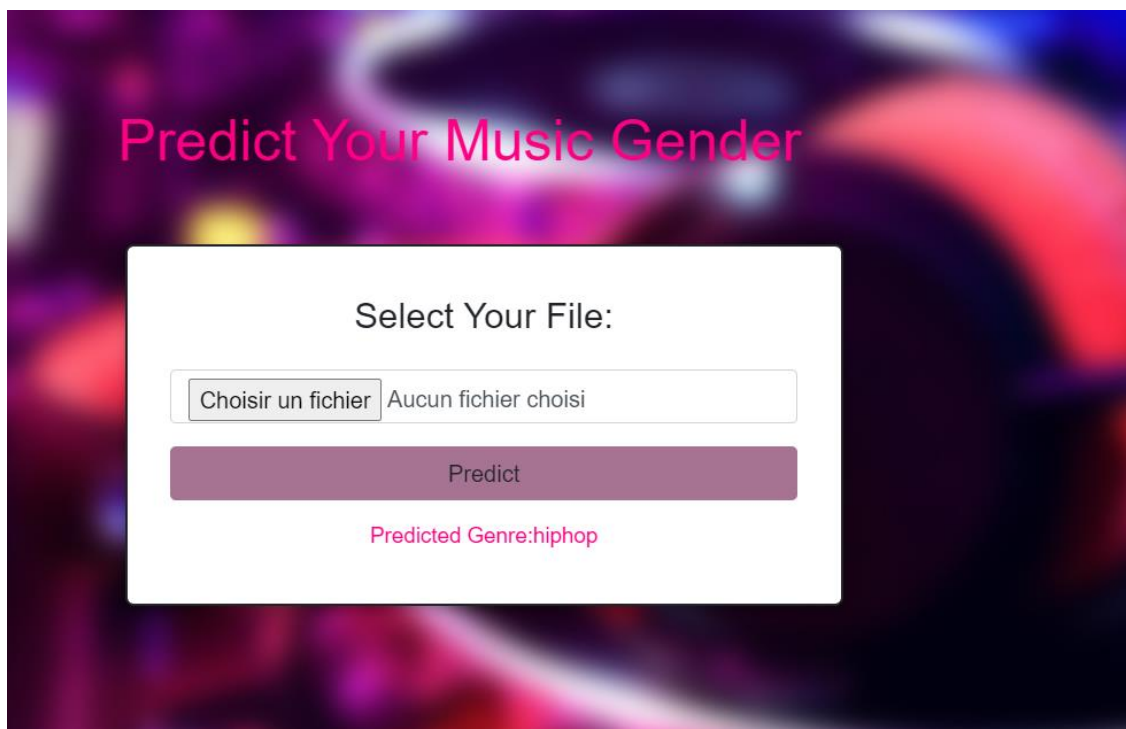


On représente par cette figure l'étape de choisir le fichier audio qui nous désirons prédire son type



**Figure 21** *Choix audio*

Cette figure présente le résultat de prédiction



**Figure 20** *Résultat de prédiction*

## Conclusion :

Dans l'ensemble, le projet démontre avec succès la création d'un environnement de Machine Learning isolé et portable en utilisant la technologie Docker. L'utilisation de conteneurs Docker pour héberger les services Flask, tels que SVM\_service et vgg19\_service, offre une solution efficace pour le déploiement et la gestion des modèles de classification musicale basés sur SVM et VGG19.

L'intégration des deux services au sein d'un même conteneur Docker, orchestré par Docker-compose, simplifie la mise en place de l'ensemble de l'application. Les services Flask agissent comme des interfaces pour recevoir des fichiers audios au format base64, les traiter, puis les transmettre aux modèles de classification correspondants.

Le recours à des volumes Docker contribue à la portabilité de l'environnement, permettant la persistance des données entre différentes instances. De plus, la déclaration explicite des dépendances dans le Dockerfile renforce la reproductibilité de l'environnement de développement.

Enfin, l'utilisation de Jenkins pour l'intégration continue et les tests ajoute une dimension cruciale au projet, assurant la qualité du code et des services déployés. En conclusion, ce projet offre une approche bien structurée et efficiente pour le déploiement d'un environnement de classification musicale basé sur le Machine Learning, avec une emphase particulière sur la portabilité, la reproductibilité, et la facilité de gestion grâce à la technologie Docker.