

1 Objectif du Projet

Le projet vise à créer un environnement Machine Learning pour la classification de genres musicaux en utilisant un dataset Kaggle. Les classifications seront effectuées à l'aide de deux modèles différents, SVM et VGG19, intégrés dans des services web Flask et déployés dans un environnement Docker.

2 Environnement de travail

1.1. Python

C'est un langage de programmation open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels.



1.2. Serveur web Flask

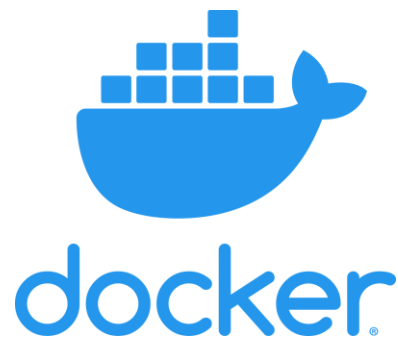
Flask est un micro framework open-source de développement web en Python. Il est classé comme micro-framework car il est très léger. Flask a pour objectif de garder un noyau simple mais extensible.



1.3. Docker

Docker est une technologie open source qui permet aux utilisateurs de créer, gérer et

déployer des applications à l'aide de conteneurs. Les applications exécutées à l'aide de Docker résident dans un environnement qui leur est propre et hébergent les dépendances nécessaires à l'exécution de l'application.



1.4. Machine Learning

Le machine learning est une technique de programmation informatique qui utilise des probabilités statistiques pour donner aux ordinateurs la capacité d'apprendre par eux-mêmes sans programmation explicite



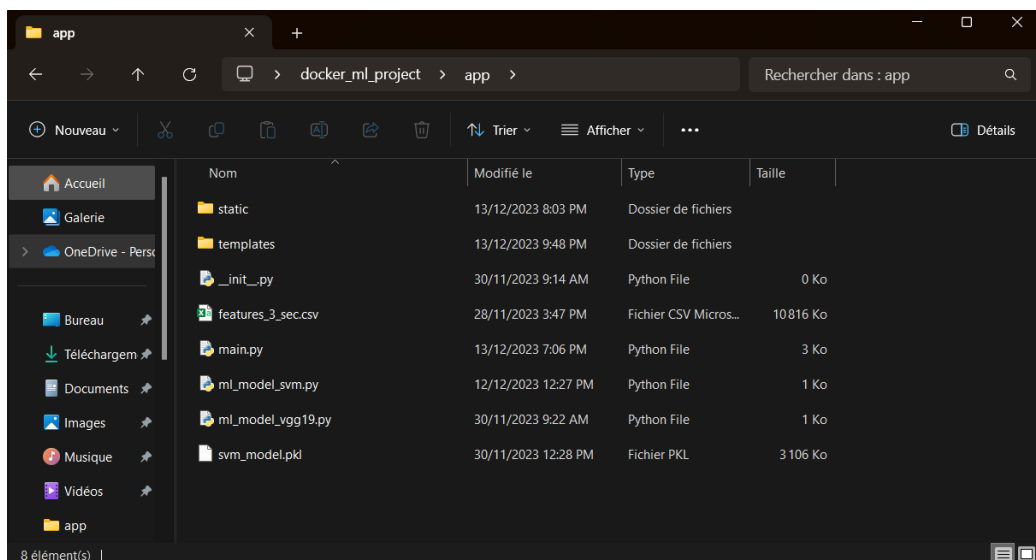
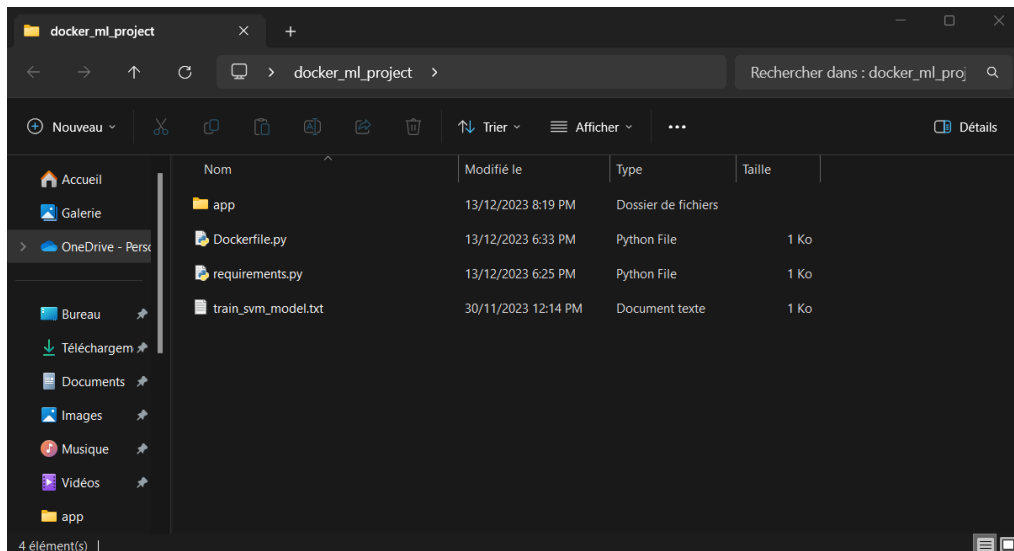
3 Réalisation et mise en œuvre

Nous arrivons maintenant à la phase de réalisation qui va mettre l'accent sur l'architecture adoptée ainsi que le bon fonctionnement du projet.

Architecture du projet

L'adoption d'une bonne architecture du projet est l'une des étapes fondamentales pour garantir la bonne organisation des fichiers.

La figure ci dessous met l'accent sur notre architecture générale.



4 Dockerfile

Dockerfile configure un environnement Python avec les dépendances nécessaires, expose un port pour l'application Flask, et définit la commande pour lancer l'application au démarrage du conteneur.

```
Fichier  Modifier  Affichage

# Utilisez une image Python comme base
FROM python:3.8-slim

# Créez le répertoire de travail et copiez les fichiers nécessaires
WORKDIR /app
COPY . /app

# Installez les dépendances
RUN apt-get update && apt-get install -y ffmpeg && rm -rf /var/lib/apt/lists/*
RUN pip install --no-cache-dir -r requirements.py

# Exposez le port sur lequel Flask va écouter
EXPOSE 5000

# Commande pour démarrer l'application Flask
CMD ["python", "app/main.py"]
```

5 Requirement

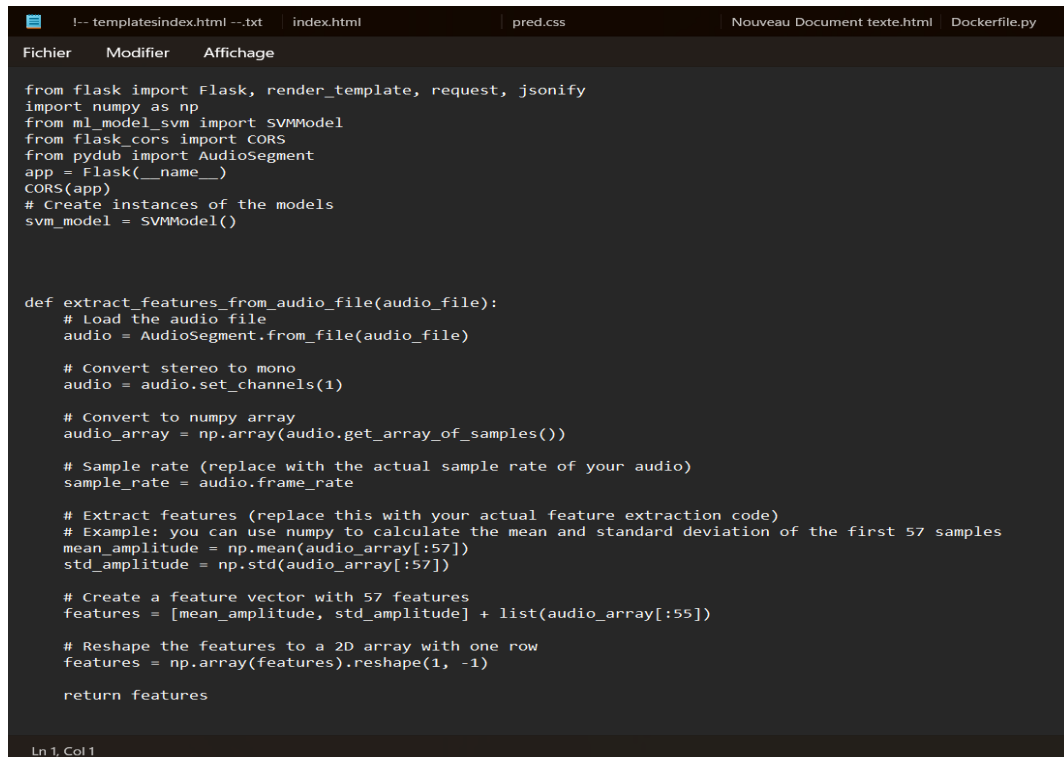
Le fichier requirements.py est un fichier de spécifications de dépendances utilisé avec l'outil pip pour installer les packages Python nécessaires à votre application. Chaque ligne indique donc une dépendance spécifique avec une version particulière. Ces versions sont souvent spécifiées pour garantir la compatibilité avec votre application et pour éviter les problèmes potentiels liés à des mises à jour futures des bibliothèques.

```
requi x + - □ x
Fichier  Modifier  Affichage  ⚙️

pandas==1.3.3
scikit-learn==0.24.2
joblib==1.3.2
Flask==2.0.1
numpy==1.21.2
Werkzeug==2.0.1
flask-cors==3.0.10
pydub==0.25.1
```

6 Main.py

Ce code est une application Flask qui fournit un service web pour la classification du genre musical en utilisant un modèle de machine learning SVM (Support Vector Machine)



```
from flask import Flask, render_template, request, jsonify
import numpy as np
from ml_model_svm import SVMModel
from flask_cors import CORS
from pydub import AudioSegment
app = Flask(__name__)
CORS(app)
# Create instances of the models
svm_model = SVMModel()

def extract_features_from_audio_file(audio_file):
    # Load the audio file
    audio = AudioSegment.from_file(audio_file)

    # Convert stereo to mono
    audio = audio.set_channels(1)

    # Convert to numpy array
    audio_array = np.array(audio.get_array_of_samples())

    # Sample rate (replace with the actual sample rate of your audio)
    sample_rate = audio.frame_rate

    # Extract features (replace this with your actual feature extraction code)
    # Example: you can use numpy to calculate the mean and standard deviation of the first 57 samples
    mean_amplitude = np.mean(audio_array[:57])
    std_amplitude = np.std(audio_array[:57])

    # Create a feature vector with 57 features
    features = [mean_amplitude, std_amplitude] + list(audio_array[:55])

    # Reshape the features to a 2D array with one row
    features = np.array(features).reshape(1, -1)

    return features
```

7 ML_model svm

Cette classe encapsule un modèle SVM pré-entraîné pour la classification du genre musical. Lorsqu'une instance de cette classe est créée, le modèle SVM est chargé. La méthode classify_genre est ensuite utilisée pour effectuer des prédictions de genre musical sur de nouvelles données en utilisant le modèle chargé.

```

Fichier  Modifier  Affichage

from sklearn import svm
import joblib
import pandas as pd

class SVMModel:
    def __init__(self):
        # Chargez le modèle SVM pré-entraîné
        self.model = joblib.load('app/svm_model.pkl')

    def classify_genre(self, features):
        # Implémentez la logique de classification ici
        prediction = self.model.predict(features)
        return prediction.tolist()

```

8 Fichier html pour le front

Ce code représente une page HTML qui crée une interface utilisateur simple pour interagir avec un service web qui prédit le genre musical d'un fichier audio en utilisant un modèle de machine learning SVM (Support Vector Machine).

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='pred.css') }}">
</head>
<body>

<div class="container">

  <header>
    <h2><a href="#"><i class="ion-plane"></i> Music </a></h2>
    <nav>
      <ul>
        <li>
          <a href="#" title="Hotels">Blues</a>
        </li>
        <li>
          <a href="#" title="Flights">Classical</a>
        </li>
        <li>
          <a href="#" title="Tours">Disco</a>
        </li>
        <li>
          <a class="btn" href="#" title="Register / Log In">Register/Log In</a>
        </li>
      </ul>
    </nav>
  </header>

  <div class="cover">
    <h1>Predict the music's genre</h1>
    <form class="flex-form" onsubmit="classifyGenre(); return false;">
      <label for="from">
        <i class="ion-location"></i>
      </label>
      <input type="file" id="audioFile" name="audio_file" accept=".wav, .mp3">
    </form>
  </div>
</div>

```

```

<input type="file" id="audioFile" name="audio_file" accept=".wav, .mp3">

    <input type="submit" value="Upload Audio">
  </form>
</div></div>
<div></div>
<div></div>
    <h1 id="result"></h1>
  </div>
</div>

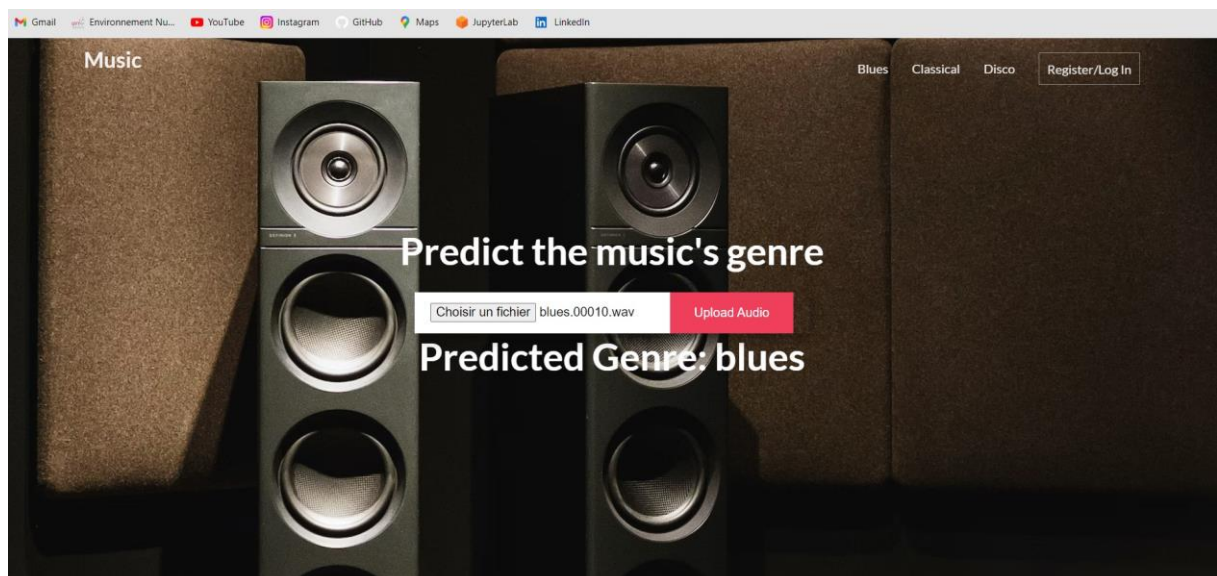
<script>
  function classifyGenre() {
    const audioFile = document.getElementById('audioFile').files[0];

    if (audioFile) {
      const formData = new FormData();
      formData.append('audio_file', audioFile);

      fetch('/svm_service', {
        method: 'POST',
        body: formData
      })
        .then(response => response.json())
        .then(data => {
          document.getElementById('result').innerText = 'Predicted Genre: ' + data.genre_prediction;
        })
        .catch(error => {
          console.error('Error:', error);
          document.getElementById('result').innerText = 'Error occurred during classification.';
        });
    } else {
      alert('Please select an audio file.');
```

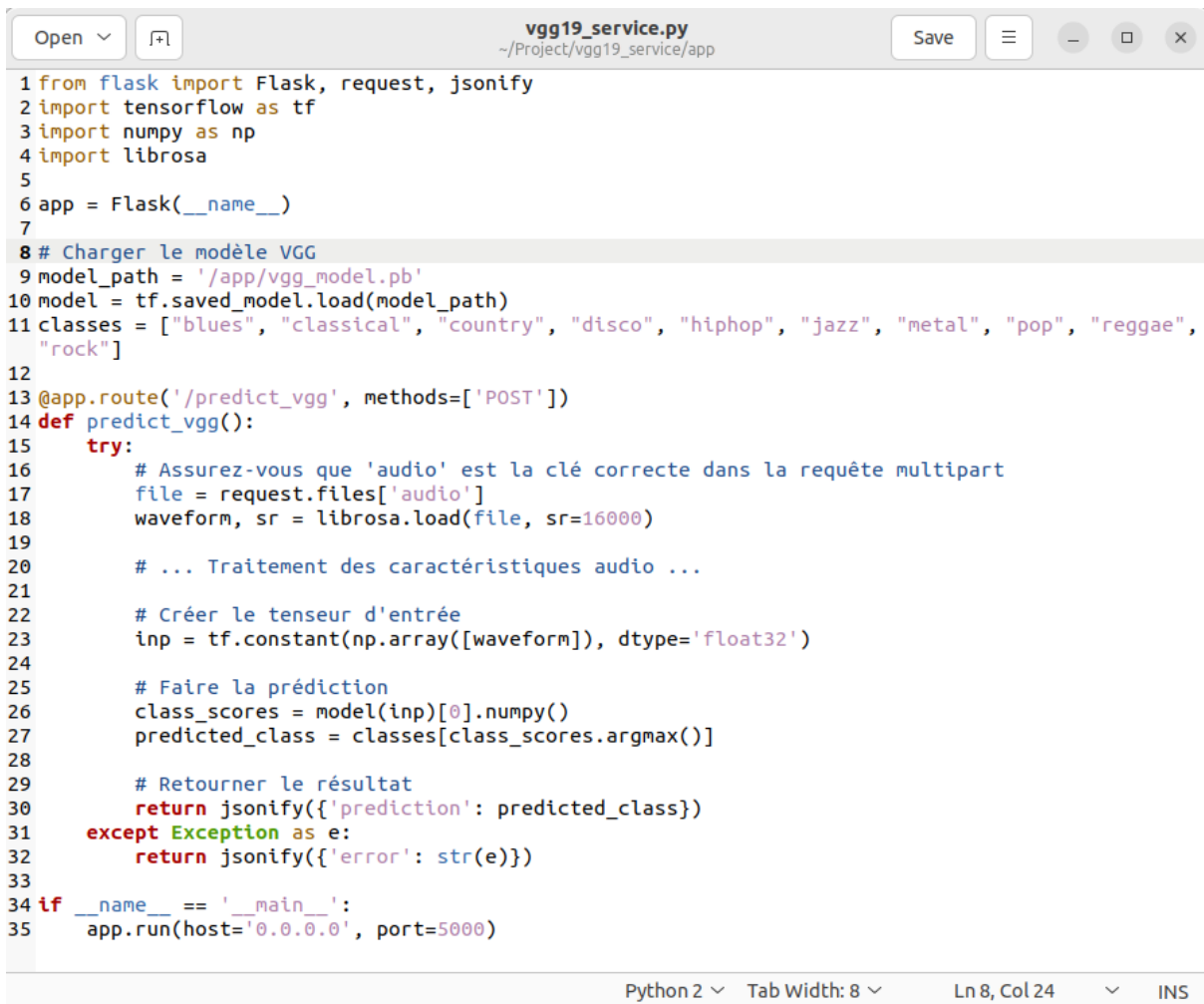
9 Interface graphique

Interface utilisateur pour télécharger un fichier audio, l'envoyer à un service web pour prédire son genre musical



10 ML_model vgg

Ce code est une application Flask qui crée un service web permettant de faire des prédictions de genre musical à l'aide d'un modèle VGG (Very Deep Convolutional Networks for Large-Scale Image Recognition).



```
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 model_path = '/app/vgg_model.pb'
10 model = tf.saved_model.load(model_path)
11 classes = ["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "pop", "reggae",
12            "rock"]
13
14 @app.route('/predict_vgg', methods=['POST'])
15 def predict_vgg():
16     try:
17         # Assurez-vous que 'audio' est la clé correcte dans la requête multipart
18         file = request.files['audio']
19         waveform, sr = librosa.load(file, sr=16000)
20
21         # ... Traitement des caractéristiques audio ...
22
23         # Créer le tenseur d'entrée
24         inp = tf.constant(np.array([waveform]), dtype='float32')
25
26         # Faire la prédiction
27         class_scores = model(inp)[0].numpy()
28         predicted_class = classes[class_scores.argmax()]
29
30         # Retourner le résultat
31         return jsonify({'prediction': predicted_class})
32     except Exception as e:
33         return jsonify({'error': str(e)})
34
35 if __name__ == '__main__':
36     app.run(host='0.0.0.0', port=5000)
```

11 docker-compose

Ce fichier docker-compose.yml définit une configuration pour un service Docker qui orchestre deux conteneurs (services) Flask, l'un pour la classification audio et l'autre pour la classification VGG19.


```
1 version: '3'
2
3 services:
4   # Flask service for audio classification
5   audio_classification:
6     build:
7       context: ./test
8     ports:
9       - "5000:5000"
10    volumes:
11      - ./test:/app
12    depends_on:
13      - vgg19_service
14    networks:
15      - my_network
16
17  # Flask service for VGG19 classification
18  vgg19_service:
19    build:
20      context: ./vgg19_service
21    ports:
22      - "5001:5000"
23    volumes:
24      - ./vgg19_service:/app
25    networks:
26      - my_network
27
28  # Add other services as needed, such as databases, etc.
29
30 networks:
31   my_network:
32     driver: bridge
33
```

12 Docker container

Cette instruction Docker vise à créer une image Docker qui intègre deux services de classification, l'un pour la classification audio et l'autre pour la classification VGG19. L'image résultante est conçue pour être exécutée comme un conteneur Docker, fournissant un environnement intégré pour accéder aux fonctionnalités des deux services de classification. L'image expose les ports 5000 et 5001, permettant ainsi aux utilisateurs d'interagir avec les services audio et VGG19 respectivement. La commande par défaut, lorsqu'un conteneur basé sur cette image est démarré, exécute simultanément les scripts des deux services, offrant ainsi une solution intégrée et prête à l'emploi pour l'utilisation conjointe des deux services de classification.

Open ▾

+

Dockerfile
~/Project

Save

≡

—

□

×

vgg19_service.py ×

docker-compose.yml ×

Dockerfile ×

```
1 # Utilisez une image de base
2 FROM python:3.8
3
4 # Définissez le répertoire de travail
5 WORKDIR /app
6
7 # Copiez les fichiers nécessaires dans l'image
8 COPY ./test /app/test
9 COPY ./vgg19_service /app/vgg19_service
10 COPY ./requirements.txt /app/requirements.txt
11
12 # Installez les dépendances
13 RUN pip install --no-cache-dir -r requirements.txt
14
15 # Exposez les ports utilisés par les services
16 EXPOSE 5000
17 EXPOSE 5001
18
19 # Commande par défaut pour exécuter les deux services
20 CMD ["sh", "-c", "python /app/test/svm_service.py & python /app/vgg19_service/vgg19_service.py"]
21
```

Dockerfile ▾

Tab Width: 8 ▾

Ln 21, Col 1 ▾

INS