**Mini Projet : Nouvelles Architectures**

Container 1 : **Frontend** :  
**Docker file :**

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory in the container

WORKDIR /app

RUN python -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Copy the requirements file into the container at /app

COPY requirements.txt .

# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Copy the entire project directory into the container at /app

COPY . .

COPY music\_svm\_model.pkl /app/models/

# Copy the start.sh script into the container at /app

COPY start.sh .

RUN chmod +x start.sh

# Expose the port Flask is running on

EXPOSE 8000

# Define environment variable for Flask

ENV FLASK\_APP=app/views.py

# Use the start.sh script as the entry point to start your application

CMD ["./start.sh"]

**Start.sh :**

#!/bin/bash

# Set environment variables if needed

export FLASK\_APP=app/views.py

export FLASK\_ENV=development

# Install Python dependencies

pip install --no-cache-dir -r requirements.txt

# Change directory to the app folder

cd /app

# Run your Flask application

flask run --host=0.0.0.0 --port=8000

**requirements.txt :**

Flask==2.0.1

Werkzeug==2.0.2

requests==2.26.0

librosa>=0.8.1

**In directory Templates:  
Index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

</head>

<body>

    <h2>Service SVM</h2>

    <form id="svmUploadForm">

        <input type="file" id="svmMusicFile" name="audioFile" accept=".wav">

        <button type="button" id="predictSVMButton">Upload and Predict (SVM)</button>

    </form>

    <div id="svmPredictionResult"></div>

    <h2>Service VGG19</h2>

    <form id="vggUploadForm">

        <input type="file" id="vggImageFile" name="imageFile" accept="image/\*">

        <button type="button" id="predictVGGButton">Upload and Predict (VGG19)</button>

    </form>

    <div id="vggPredictionResult"></div>

    <script>

    document.getElementById('predictSVMButton').addEventListener('click', function() {

        const fileInput = document.getElementById('svmMusicFile');

        const file = fileInput.files[0];

        if (!file) {

            console.error('No file selected!');

            return;

        }

        const formData = new FormData();

        formData.append('audioFile', file);

        fetch('/make\_prediction', {

            method: 'POST',

            body: formData,

        })

        .then(response => response.json())

        .then(data => {

            console.log(data);

            const predictionResult = document.getElementById('svmPredictionResult');

            predictionResult.innerHTML = `Predicted genre: ${data.prediction}`;

        })

        .catch(error => {

            console.error('Error:', error);

            // Handle error

        });

    });

    document.getElementById('predictVGGButton').addEventListener('click', function() {

        const fileInput = document.getElementById('vggImageFile');

        const file = fileInput.files[0];

        if (!file) {

            console.error('No file selected!');

            return;

        }

        const formData = new FormData();

        formData.append('imageFile', file);

        fetch('/predict\_vgg19', {

            method: 'POST',

            body: formData,

        })

        .then(response => response.json())

        .then(data => {

            console.log(data);

            const predictionResult = document.getElementById('vggPredictionResult');

            predictionResult.innerHTML = `Predicted genre: ${data.prediction}`;

        })

        .catch(error => {

            console.error('Error:', error);

            // Handle error

        });

    });

    </script>

</body>

</html>

**Views.py:**

import os

from flask import Flask, request, render\_template, jsonify

import requests

import librosa

from sklearn.preprocessing import StandardScaler

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

    return render\_template('index.html')

@app.route('/make\_prediction', methods=['POST'])

def make\_prediction():

    try:

        file = request.files['audioFile']

        print(f"Received file: {file.filename}")

        mfccs\_scaled\_features = preprocess\_audio(file)

        if mfccs\_scaled\_features is None:

            return jsonify({'error': 'Failed to preprocess audio'})

        response = send\_prediction\_request(mfccs\_scaled\_features.tolist())

        return jsonify(response)

    except Exception as e:

        return jsonify({'error': str(e)})

def preprocess\_audio(file):

    try:

        file\_path = 'temp\_audio.wav'

        file.save(file\_path)

        audio, sample\_rate = librosa.load(file\_path, sr=None)  # Load audio with original sample rate

        os.remove(file\_path)  # Remove the temporary file

        mfccs\_features = librosa.feature.mfcc(y=audio, sr=sample\_rate, n\_mfcc=57)

        scaler = StandardScaler()

        mfccs\_scaled\_features = scaler.fit\_transform(mfccs\_features)

        mfccs\_scaled\_features = mfccs\_scaled\_features.reshape(1, -1)

        return mfccs\_scaled\_features

    except Exception as e:

        print(f"Error in audio preprocessing: {str(e)}")

        return None

def send\_prediction\_request(features):

    try:

        svm\_service\_url = 'http://svm-service:5000/predict'

        response = requests.post(svm\_service\_url, json={'features': features})

        if response.status\_code == 200:

            return response.json()

        else:

            return {'error': 'Prediction failed'}

    except Exception as e:

        return {'error': str(e)}

@app.route('/predict\_vgg19', methods=['POST'])

def predict\_vgg19():

    try:

        file = request.files['imageFile']

        print(f"Received file: {file.filename}")

        # Process image file and send to VGG19 service on port 6000

        vgg\_service\_url = 'http://vgg19-service:6000/predict\_vgg19'  # Change this to your VGG19 service endpoint

        files = {'file': file.read()}  # Read file content

        response = requests.post(vgg\_service\_url, files=files)

        if response.status\_code == 200:

            prediction = response.json().get('prediction', 'Unknown')

            return jsonify({'prediction': prediction})

        else:

            return jsonify({'error': 'Failed to get prediction from VGG19 service'})

    except Exception as e:

        return jsonify({'error': str(e)})

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host='0.0.0.0', port=8000, debug=True)

**Test:**

import io

import unittest

from app.views import app

class FrontendTests(unittest.TestCase):

    def setUp(self):

        self.app = app.test\_client()

    def test\_file\_upload(self):

        # Simulate file upload and test if the file is properly sent to the backend

        data = {'file': (io.BytesIO(b'my file contents'), 'Data/genres\_original/blues/blues.00000.wav')}

        response = self.app.post('/make\_prediction', data=data, content\_type='multipart/form-data')

        self.assertEqual(response.status\_code, 200)

Container 2 : **Service\_SVM** :

**Docker file :**

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory in the container

WORKDIR /app

RUN python -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Copy the requirements file into the container at /app

COPY requirements.txt .

# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Copy the entire project directory into the container at /app

COPY . .

COPY music\_svm\_model.pkl /app/models/

# Copy the start.sh script into the container at /app

COPY start.sh .

RUN chmod +x start.sh

# Expose the port Flask is running on

EXPOSE 5000

# Define environment variable for Flask

ENV FLASK\_APP=app/views.py

# Use the start.sh script as the entry point to start your application

CMD ["./start.sh"]

**Start.sh:**

#!/bin/bash

# Set environment variables if needed

export FLASK\_APP=app/views.py

export FLASK\_ENV=development

# Install Python dependencies

pip install --no-cache-dir -r requirements.txt

# Run your Flask application

flask run --host=0.0.0.0 --port=5000

**Requirement.txt:**

Flask==2.0.1

Werkzeug==2.0.2

scikit-learn==0.24.2

numpy>=1.22

librosa>=0.8.1

pandas==1.3.4

joblib==1.1.0

seaborn==0.11.2

flask-cors>=3.1.1

**In app :   
 GenreMusicClassification.py**

import pandas as pd  
from sklearn import svm

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.model\_selection import train\_test\_split

import joblib

import os

data = pd.read\_csv('Data/features\_3\_sec.csv')

X = data.iloc[:, 1:-1]  # Features

y = data['label']  # Target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# convert scaled arrays back to DataFrame

X\_train = pd.DataFrame(X\_train\_scaled, columns=X.columns)

X\_test = pd.DataFrame(X\_test\_scaled, columns=X.columns)

svm\_classifier = svm.SVC(kernel='linear', C=1.0)

svm\_classifier.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = svm\_classifier.predict(X\_test)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

joblib.dump(svm\_classifier, 'models/music\_svm\_model.pkl')

# Print the evaluation metrics

print(f'Accuracy: {accuracy}')

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1 Score: {f1}')

# Load the saved model

model\_path = 'models/music\_svm\_model.pkl'

loaded\_model = joblib.load(model\_path)

# Use the loaded model to make predictions

sample\_features = X\_test.iloc[0:1]  # This should work now as X\_test is a DataFrame

sample\_features\_scaled = scaler.transform(sample\_features)

sample\_features\_scaled\_df = pd.DataFrame(sample\_features\_scaled, columns=sample\_features.columns)

prediction = loaded\_model.predict(sample\_features\_scaled\_df)

print(f"Predicted value: {prediction}")

**And views.py:**

from flask import Flask, request, jsonify  
from sklearn.preprocessing import StandardScaler

import joblib

import numpy as np

import librosa

import os

app = Flask(\_\_name\_\_)

# Inside the predict function in your SVM service

@app.route('/predict', methods=['POST'])

def predict():

    try:

        features = request.json.get('features')

        if features is None:

            return jsonify({'error': 'Features not found in request'})

        # Load the model

        model\_path = 'models/music\_svm\_model.pkl'

        if os.path.exists(model\_path):

            loaded\_model = joblib.load(model\_path)

            prediction = loaded\_model.predict(features)

            if prediction is None or prediction[0] == 'undefined':

                return jsonify({'error': 'Prediction not available!'})

        return jsonify({'error': 'Model file not found!'})

    except Exception as e:

        return jsonify({'error': str(e)})

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

**Test:**

import unittest import pandas as pd

from app.views import app

class SVMServiceTests(unittest.TestCase):

    def setUp(self):

        self.app = app.test\_client()

    def test\_prediction\_with\_valid\_data(self):

        # Mocking the SVM service response:

        svm\_data = pd.read\_csv('Data/features\_3\_sec.csv')

        X = svm\_data.iloc[:, 1:-1]  # Features

        y = svm\_data['label']   # Example data for the prediction

        # Send a POST request with JSON data

        response = self.app.post('/predict', json={'features': X.to\_dict(), 'labels': y.tolist()})

        # Assertions

        self.assertEqual(response.status\_code, 200)

        response\_data = response.get\_json()  # Get JSON data from response

        self.assertIn('prediction', response\_data)  # Check if 'prediction' key exists in the response

        # Add further assertions based on your service's expected behavior

    def test\_prediction\_with\_invalid\_data(self):

        # Test with invalid data (empty data, wrong format, etc.)

        invalid\_data = {}  # Example of invalid data

        # Send a POST request with invalid JSON data

        response = self.app.post('/predict', json=invalid\_data)

        # Assertions

        self.assertEqual(response.status\_code, 400)  # Assuming 400 for an invalid request

        # Add assertions for the error response or message returned by your service

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

Container 3 : **Service\_VGG19** :

**Docker file :**

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory in the container

WORKDIR /app

RUN python -m venv /opt/venv

ENV PATH="/opt/venv/bin:$PATH"

# Copy the requirements file into the container at /app

COPY requirements.txt .

# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Copy the entire project directory into the container at /app

COPY . .

COPY music\_genre\_vgg19\_model.pkl /app/models

# Copy the start.sh script into the container at /app

COPY start.sh .

RUN chmod +x start.sh

# Expose the port Flask is running on

EXPOSE 6000

# Define environment variable for Flask

ENV FLASK\_APP=app/views.py

# Use the start.sh script as the entry point to start your application

CMD ["./start.sh"]

**Start.sh:**

#!/bin/bash

# Set environment variables if needed

export FLASK\_APP=app/views.py

export FLASK\_ENV=development

# Install Python dependencies

pip install --no-cache-dir -r requirements.txt

# Run your Flask application

flask run --host=0.0.0.0 --port=6000

**requirements.txt:**

Flask==2.0.1

numpy==1.19.5

scikit-learn==0.24.2

tensorflow==2.7.0

Werkzeug==2.0.2

requests==2.26.0

librosa>=0.8.1

protobuf==3.20.0

**vggClassification.py**

import os

import numpy as np

from tensorflow.keras.preprocessing import image

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.applications import VGG19

from tensorflow.keras.optimizers import Adam

import pickle

# Chemin vers le dossier contenant les images

data\_path = 'Data/images\_original'

# Liste des genres musicaux

genres = os.listdir(data\_path)

# Chargement des images et des labels

images = []

labels = []

for genre\_idx, genre in enumerate(genres):

    genre\_path = os.path.join(data\_path, genre)

    for img\_name in os.listdir(genre\_path):

        img\_path = os.path.join(genre\_path, img\_name)

        img = image.load\_img(img\_path, target\_size=(224, 224))

        img\_array = image.img\_to\_array(img)

        images.append(img\_array)

        labels.append(genre\_idx)

# Conversion en tableau numpy

images = np.array(images)

labels = np.array(labels)

# Séparation des données en ensembles d'entraînement et de validation

X\_train, X\_val, y\_train, y\_val = train\_test\_split(images, labels, test\_size=0.2, random\_state=42)

# Charger le modèle VGG19 pré-entraîné

vgg\_base = VGG19(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Ajouter des couches fully connected pour la classification des genres musicaux

model = Sequential()

model.add(vgg\_base)

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dense(len(genres), activation='softmax'))  # Nombre de classes = nombre de genres

# Geler les poids des couches du modèle VGG de base pour ne pas les ré-entraîner

for layer in vgg\_base.layers:

    layer.trainable = False

# Compiler le modèle

model.compile(optimizer=Adam(lr=0.0001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Entraîner le modèle

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_val, y\_val))

# Sauvegarder le modèle

with open('models/music\_genre\_vgg19\_model.pkl', 'wb') as file:

    pickle.dump(model, file)

**views.py:**

from flask import Flask, request, jsonify

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.vgg19 import preprocess\_input, decode\_predictions

from tensorflow.keras.optimizers import Adam

import numpy as np

import base64

import io

import pickle

app = Flask(\_\_name\_\_)

# Charge le modèle spécifique pour la classification des genres musicaux

try:

    with open('music\_genre\_vgg19\_model.pkl', 'rb') as file:

        model = pickle.load(file)

except Exception as e:

    print("Error loading the model:", e)

@app.route('/predict\_vgg19', methods=['POST'])

def predict\_vgg19():

    try:

        file = request.files['imageFile']

        print(f"Received file: {file.filename}")

        # Image preprocessing for VGG19

        decoded\_image = file.read()

        img = image.img\_to\_array(image.load\_img(io.BytesIO(decoded\_image), target\_size=(224, 224)))

        img = np.expand\_dims(img, axis=0)

        img = preprocess\_input(img)

        # Perform predictions with the loaded VGG19 model

        preds = model.predict(img)

        decoded\_preds = decode\_predictions(preds, top=1)[0]

        predicted\_genre = decoded\_preds[0][1]  # Assuming the second element in the prediction tuple is the genre

        return jsonify({'prediction': predicted\_genre})

    except Exception as e:

        return jsonify({'error': str(e)})

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, host='0.0.0.0')

**test:**

import os

import unittest

from app.views import app

class VGGServiceTests(unittest.TestCase):

    def setUp(self):

        self.app = app.test\_client()

    def test\_prediction(self):

        # Path to the directory containing the images

        data\_path = 'Data/images\_original'

        # List of music genres

        genres = os.listdir(data\_path)

        try:

            response = self.app.post('/predict\_vgg19', json={'image\_base64': 'base64\_encoded\_image'})

            self.assertEqual(response.status\_code, 200)

            # Assert other conditions based on your service's expected behavior

        except Exception as e:

            print(f"Error occurred: {str(e)}")

            # Handle the error gracefully

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Docker-compose:**

version: '3'

services:

  svm-service:

    build: ./FluskProjectBackendSVM

    ports:

      - "5000:5000"

    volumes:

      - shared-models:/app/models  # Utilisation d'un volume nommé partagé pour les modèles

  frontend-service:

    build: ./FluskProjectFrontend

    ports:

      - "8000:8000"

    volumes:

      - shared-models:/app/models  # Utilisation du même volume nommé pour les modèles

  vgg19-service:

    build: ./FluskProjectBackendVGG

    ports:

      - "6000:6000"

volumes:

      - shared-models:/app/models  # Utilisation du même volume nommé pour les modèles

  tests:

    build: ./tests

    depends\_on:

      - frontend-service

      - svm-service

      - vgg19-service

volumes:

  shared-models:  # Volume nommé partagé pour les modèles

**Une image contenant texte, logiciel, Icône d’ordinateur, Page web

Description générée automatiquement**

Une image contenant texte, logiciel, Icône d’ordinateur, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, logiciel, Icône d’ordinateur, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

Une image contenant texte, logiciel, Icône d’ordinateur, Logiciel multimédia

Description générée automatiquement

pipeline {

    agent any

    stages {

        stage('Build and Test Frontend') {

            steps {

                dir('frontend') {

                    sh 'docker build -t frontend-service .'

                    sh 'docker run --rm frontend-service npm test'

                }

            }

        }

        stage('Build and Test SVM Service') {

            steps {

                dir('svm\_service') {

                    sh 'docker build -t svm-service .'

                    sh 'docker run --rm svm-service pytest -v test\_svm\_service.py'

                }

            }

        }

        stage('Build and Test VGG Service') {

            steps {

                dir('vgg\_service') {

                    sh 'docker build -t vgg-service .'

                    sh 'docker run --rm vgg-service pytest -v test\_vgg\_service.py'

                }

            }

        }

        stage('Deployment') {

            steps {

                sh 'docker-compose up -d --build'

            }

        }

    }

    post {

        always {

            sh 'docker-compose down'

        }

        success {

            echo 'Pipeline succeeded!'

        }

        failure {

            echo 'Pipeline failed!'

        }

    }

}

Une image contenant texte, capture d’écran, logiciel, ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, ordinateur

Description générée automatiquement