Classification des genres musicaux - Rapport du projet

Présenté par : Mohamed Iheb BEN ROMDHANE
Classe : 3ème année génie Informatique NTS
Année universitaire : 2023/2024

1 Introduction

Le projet a été élaboré dans le dessein d'exploiter les avantages offerts par Docker en matière de flexibilité et de portabilité. En optant pour cette technologie, notre objectif était de créer un environnement spécialisé en apprentissage automatique (Machine Learning ou ML), axé sur la classification des genres musicaux. Le choix du dataset GTZAN, renommé pour sa diversité et son importance dans le domaine musical, découle de notre ambition de développer des modèles robustes capables de s'adapter à une vaste gamme de styles musicaux.

L'adoption de Python comme principal langage de programmation nous a permis de tirer parti de sa popularité et des nombreuses bibliothèques spécialisées dans le domaine de l'apprentissage automatique. Pour la création d'une interface web, nous avons opté pour Flask en raison de sa légèreté et de sa facilité d'intégration avec les applications de ML. Enfin, les modèles SVM (Support Vector Machine) et VGG19 ont été choisis en raison de leurs performances éprouvées dans la classification musicale.

Le but global de ce projet est de rationaliser le processus de déploiement et de gestion des services en ML, en proposant une solution à la fois efficace et facilement transférable d'un environnement à un autre.

2 Objectifs du Projet

Services de Classification :

- **svm_service :** Ce service utilise un modèle basé sur SVM pour effectuer la classification des genres musicaux.
- **vgg19_service :** Ce service fait appel à un modèle VGG19 pour réaliser la classification musicale.

Base de Données d'Entraînement : Le modèle de Machine Learning est entraîné sur le jeu de données GTZAN, accessible sur Kaggle via le lien suivant : https ://www.kaggle.com/andradaolteanu dataset-music-genre-classification. Cet dataset offre une diversité de genres musicaux, contribuant ainsi à la robustesse du modèle.

Déploiement Simplifié : Le projet vise à simplifier le déploiement en utilisant Docker et Docker-compose. Ces outils permettent de créer une image unique encapsulant les deux services de classification (svm_service et vgg19_service), facilitant ainsi le déploiement uniforme de l'application sur différentes plates-formes.

Intégration Continue avec Jenkins : Pour assurer la qualité continue du projet, un environnement d'intégration continue est mis en place avec Jenkins. Cela permet d'automatiser le déploiement et les tests, garantissant ainsi une gestion efficace des mises à jour et des évolutions du code source.

3 Environnement Machine Learning

3.1 Choix du Dataset

GTZAN a été choisi pour son ampleur, sa variété musicale équilibrée et sa renommée dans la recherche en classification musicale, en faisant ainsi un choix idéal pour former des modèles généralisables à divers genres musicaux.

3.2 Modèles de Machine Learning

Les choix des modèles SVM et VGG19 résultent de leurs compétences spécifiques. SVM excelle dans la classification de données de grande dimension, le rendant ainsi approprié pour la

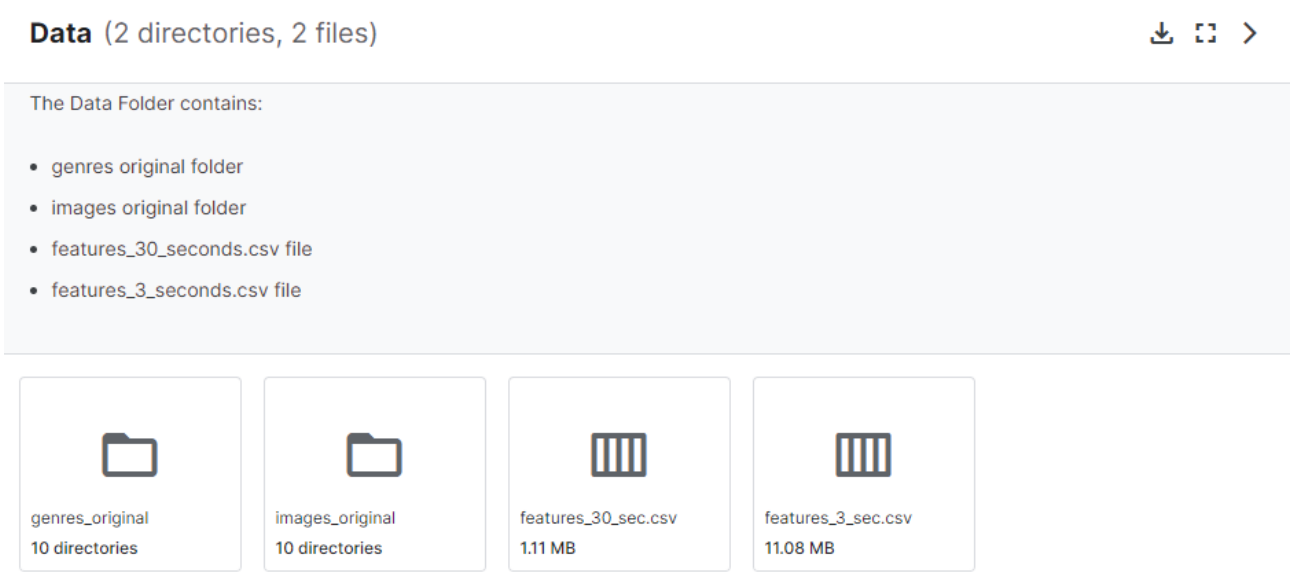

FIGURE 1 – Dataset GTZAN

classification musicale basée sur des caractéristiques extraites des fichiers audio. En revanche, VGG19, en tant que réseau neuronal convolutif profond, est apte à saisir des motifs complexes et à extraire des représentations hiérarchiques, ce qui le rend particulièrement adapté à la reconnaissance de motifs dans les signaux audio.

3.3 Services Web Flask

3.3.1 SVM_service

Le service SVM a été créé avec le framework Flask afin de fournir une API RESTful. Cette API prend en charge la classification du genre musical pour un fichier audio au format wav encodé en base64. Lorsqu'une requête est reçue, le service décode le fichier, réalise la prédiction en utilisant le modèle SVM pré-entraîné, et renvoie le résultat au format JSON.

```python
@app.route('/uploaderSVM', methods=['POST'])
def upload_file():
    if request.method == 'POST':
        audio_base64 = request.form['audio_base64']

        audio_data = base64.b64decode(audio_base64)
        filename = 'uploaded_audio.wav'
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)

        with open(filepath, 'wb') as f:
            f.write(audio_data)

        signal, rate = librosa.load(filepath)
        S = librosa.feature.melspectrogram(signal, sr=rate, n_fft=2048, hop_length=512, n_mels=128)
        S_DB = librosa.power_to_db(S, ref=np.max)
        S_DB = S_DB.flatten()[:1200]

        # Load the SVM model
        clf = pickle.load(open('SVM.pkl', 'rb'))

        # Make predictions
        ans = clf.predict([S_DB])[0]
        music_class = str(ans)

        print(music_class)
        return music_class
```

FIGURE 2 – code du service SVM

3.3.2 VGG19_service

Le service VGG19 constitue une extension du concept établi par le service SVM, avec une intégration spécifique du modèle VGG19 pour la classification des genres musicaux à partir de fichiers audio encodés en base64. Bien que le service VGG19 partage des similitudes fondamentales avec le service SVM en termes d'architecture Flask et de manipulation de fichiers audio encodés, sa distinction réside dans l'utilisation d'un modèle de réseau neuronal profond VGG19 préentraîné pour la classification.

```python
def image_to_base64(image_path):
    with open(image_path, "rb") as image_file:
        encoded_image = base64.b64encode(image_file.read())
        return encoded_image.decode("utf-8")

@app.route('/uploaderVGG19', methods=['POST'])
def classify_image():
    data = request.get_json()

    if data and "image_data" in data:
        encoded_image_data = data["image_data"]
        decoded_image_data = base64.b64decode(encoded_image_data)

        temp_image_file = '/3eme/NouvArchi/temp_image.jpg'
        with open(temp_image_file, 'wb') as temp_file:
            temp_file.write(decoded_image_data)

        img = image.load_img(temp_image_file, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255.0


        predictions = model.predict(img_array)

        genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
        predicted_genre = genres[np.argmax(predictions)]

        response_data = {"received_message": "Image file received and processed successfully",
                         "response": f"Predicted Genre: {predicted_genre}"}
    else:
        response_data = {"received_message": "No image file received", "response": "Error"}

    return jsonify(response_data)
```

FIGURE 3 – code du service VGG19

3.4 Docker Container et Docker-Compose

3.4.1 Structure du Docker Container

Le conteneur Docker joue un rôle central dans l'architecture du projet en encapsulant de manière complète et efficiente l'environnement de Machine Learning (ML). Cette encapsulation revêt une importance cruciale pour garantir la portabilité du système, facilitant ainsi le déploiement uniforme de l'application sur différentes machines et dans divers environnements.

```
FROM python:3.8-slim-buster

EXPOSE 5000

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

# Install pip requirements
COPY ./requirements.txt /var/www/App/requirements.txt
RUN python -m pip install -r /var/www/App/requirements.txt

RUN pip install --ignore-installed six watson-developer-cloud
RUN pip install soundfile
WORKDIR /app
COPY . /app
ENV NUMBA_CACHE_DIR=/tmp/numba_cache


RUN useradd appuser && chown -R appuser /app
USER appuser

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

FIGURE 4 – Dockerfile pour le contenaire de backend

```dockerfile
FROM python:3.8-slim-buster

EXPOSE 8080

# Keeps Python from generating .pyc files in the container
ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

# Install pip requirements
COPY requirements.txt .
RUN pip install -r requirements.txt

WORKDIR /app
COPY . /app

RUN useradd appuser && chown -R appuser /app
USER appuser

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

FIGURE 5 – Dockerfile pour le contenaire de frontend

3.4.2 Docker-Compose

Le fichier Docker-Compose représente un élément central de l'infrastructure de déploiement de notre projet. Il offre une solution simplifiée pour orchestrer les services Flask, gérer les dépendances, et exposer les ports nécessaires afin de faciliter la communication entre les divers composants de l'application. Cette approche est alignée avec la philosophie de Docker, simplifiant considérablement le processus de déploiement et de gestion de l'ensemble de l'application.

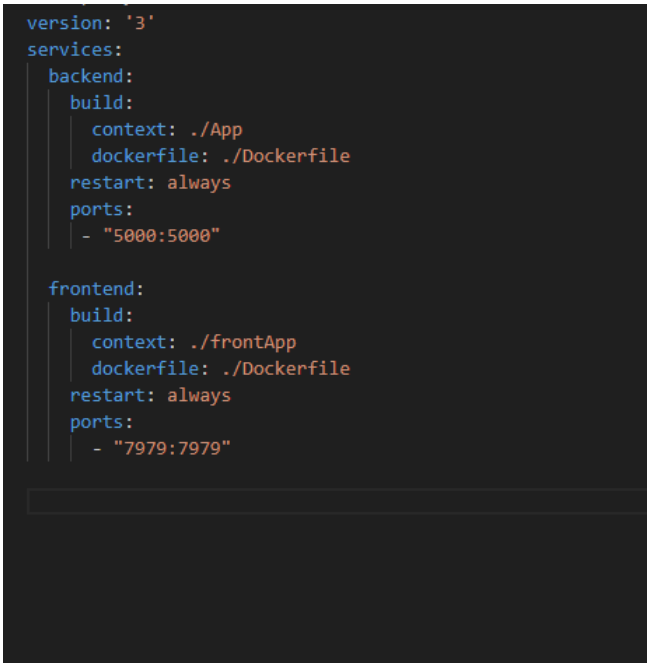

FIGURE 6 – code du fichier docker-compose.yml

4 Jenkins

Jenkins est configuré pour surveiller le référentiel du code source. À chaque modification, Jenkins déclenche une série de tests automatisés, y compris des tests unitaires pour les services Flask, ainsi que des tests d'intégration pour garantir la communication correcte entre les services.

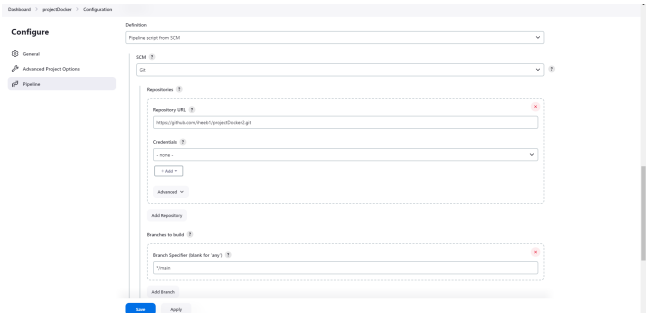

FIGURE 7 – Enter Caption

```
pipeline {
    agent any

    environment {
        PYTHON_VERSION = '3.8'
    }

    stages {
        stage('Build') {
            steps {
                script {
                    bat "docker-compose build"
                }
            }
        }

        stage("Test") {
            steps {
                script {
                    bat "py unittests.py"
                }
            }
            post {
                always {
                    junit 'test-reports/*.xml'
                }
            }
        }

        stage("Run") {
            steps {
                script {
                    bat "docker-compose up"
                }
            }
        }
    }
}
```

FIGURE 8 – code du fichier jenkinsfile

5 Les Tests Unitaires

Les tests unitaires jouent un rôle essentiel dans l'assurance de la qualité de notre application ML basée sur Docker. Dans le cadre de ce projet, deux suites de tests ont été élaborées à l'aide du module unittest de Python, visant à évaluer la fonctionnalité des services Flask dédiés aux modèles SVM et VGG19. Ces tests émettent des requêtes GET aux points d'extrémité respectifs, simulant ainsi les interactions utilisateur. Les assertions effectuées vérifient non seulement le statut des réponses, assurant qu'elles sont "200 OK", mais vont également au-delà en inspectant le contenu des réponses JSON retournées. Cette approche garantit que les réponses des services incluent les données attendues, renforçant ainsi la fiabilité de l'ensemble du système. L'intégration de Jenkins dans le processus de développement automatise l'exécution régulière de ces tests, contribuant à une détection rapide des éventuels problèmes et à la préservation de la stabilité du projet.

```python
import unittest
import app
import json

BASE_URL = 'http://127.0.0.1:5000/'

class TestFlaskApi(unittest.TestCase):

    def setUp(self):
        self.app = app.app.test_client()
        self.app.testing = True

    def test_svm(self):
        rv = self.app.get(BASE_URL + 'uploaderSVM')
        self.assertEqual(rv.status, '200 OK')
        response_data = json.loads(rv.get_data(as_text=True))
        self.assertIn('genre', response_data)

    def test_vgg(self):
        rv = self.app.get(BASE_URL + 'uploaderVGG19')
        self.assertEqual(rv.status, '200 OK')
        response_data = json.loads(rv.get_data(as_text=True))
        self.assertIn('genre', response_data)

if __name__ == '__main__':
    import xmlrunner
    runner = xmlrunner.XMLTestRunner(output='test-reports')
    unittest.main(testRunner=runner)
```

FIGURE 9 – code de fichier unittests.py

FIGURE 10 – Build avec succés

6 Conclusion Générale

Ce projet représente une réussite remarquable en intégrant efficacement Docker dans le domaine complexe du Machine Learning, axé sur la classification des genres musicaux. L'utilisation de Python, Flask, et des modèles SVM et VGG19 a permis de créer un environnement ML robuste et flexible.

L'encapsulation complète dans un conteneur Docker garantit la portabilité et la reproductibilité de l'application sur différentes machines. Le fichier Docker-Compose simplifie le déploiement en orchestrant élégamment les services Flask, facilitant ainsi la gestion des dépendances et des ports.

L'intégration de Jenkins en tant que système d'intégration continue assure la stabilité du projet en automatisant les tests, notamment des tests unitaires pour les services Flask. Cette approche contribue à une maintenance efficace de l'application.

En résumé, ce projet démontre avec succès la conception et le déploiement d'un environnement ML basé sur Docker, soulignant la puissance de cette approche pour rendre la gestion et le déploiement des applications ML plus accessibles et efficaces. Les enseignements tirés ouvrent la voie à des développements futurs dans le domaine passionnant de la classification musicale et de l'intégration de technologies innovantes.