

Rapport de mini Projet

Par

Sourour METHNI

Prédiction de genre musical d'un fichier Wav par svm et vgg19

Enseignant

BOULAARES Mehrez

Année universitaire : 2022-2023

Table des matières

Introduction	1
1. Présentation de l'environnement logiciel	1
2. Architecture de mon projet.....	2
3. Service svm	2
4. Service vgg19.....	4
5. Dockerfile pour le front-end.....	6
6. Dockerfile pour svm.....	7
7. Dockerfile pour vgg19	7
8. Communication entre les trois conteneurs	8
9. Tester avec Docker.....	8
Conclusion.....	10

Table des figures

Figure 1- Logo de Visual studio code	1
Figure 2- Logo de python.....	1
Figure 3- Logo de Docker	2
Figure 4- Architecture de mon projet	2
Figure 5- Capture de service svm.....	3
Figure 6- Capture de service vgg19	5
Figure 7- Prediction avec svm.....	5
Figure 8- Dockerfile de la partie front-end	6
Figure 9- Front-end de mon projet	6
Figure 10- Dockerfile pour le service svm.....	7
Figure 11- Dockerfile pour le service vgg19	7
Figure 12- Exécution de docker-compose.....	8
Figure 13- Capture de Jenkinsfile	9
Figure 14- Fichier de tests	9

Introduction

Ce projet vise à créer un environnement de Machine Learning basé sur Python et le serveur web Flask, permettant de classifier les genres musicaux à partir de dataset GTZAN Music Genre Classification disponible sur Kaggle.

Le projet comprend la mise en place de deux services web Flask chacun dans un conteneur Docker l'un effectue cette classification à l'aide de modèle SVM et l'autre avec le modèle VGG19.

La communication entre les deux conteneurs est réalisée à l'aide de docker-compose.

En effet, à chaque conteneur on attribue une adresse IP et se lance sur un port spécifique.

1. Présentation de l'environnement logiciel

Pour réaliser ce mini projet j'ai utilisé différents logiciel et langages de programmation que je vais les détailler dans cette partie.

Pour coder j'ai utilisé Visual studio code qui est un éditeur de code personnalisable, léger et supportant de nombreux langages. La Figure 1 présente le logo de cet outil.



Figure 1- Logo de Visual studio code

Pour développer les deux services web Flask j'ai utilisé Python qui est un langage de programmation polyvalent, simple lisible et populaire. La Figure 2 illustre le logo de python.



Figure 2- Logo de python

Pour appliquer le principe de conteneurisation j'ai utilisé la plateforme de conteneurisation pour créer, distribuer et exécuter des applications de manière portable et isolée. La Figure 3 montre le logo de Docker.



Figure 3- Logo de Docker

2. Architecture de mon projet

L'architecture de mon projet est présentée par la Figure 4.

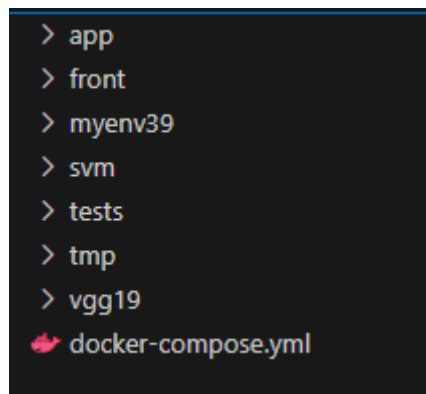


Figure 4- Architecture de mon projet

3. Service svm

C'est un service qui permet de prédire le genre musical d'un fichier Wav passé en paramètre en suivant les étapes suivantes :

- Chargement de modèle
- Régler la route Endpoint Flask pour la prédiction avec svm : /svm
- Exécuter la fonction `predict_genre_svm()`
- Exécution de l'application Flask : `If __name__ == '__main__': app.run(host='0.0.0.0', port=510)`

Pour la création de modèle svm j'ai suivi dans mon code les étapes suivantes :

- ❖ Chargement des données : à partir d'un fichier CSV sur Kaggle
- ❖ Division des données : Les données sont séparées en ensembles d'entraînement et de test à l'aide de `train_test_split`.
- ❖ Réduction de dimensionnalité avec PCA : Il applique l'analyse en composantes principales (PCA) sur les données d'entraînement pour réduire leur dimensionnalité.

- ❖ Construction du modèle SVM : Un modèle de machine à vecteurs de support (SVM) avec un noyau linéaire est créé.
- ❖ Transformation des données avec PCA : Les données d'entraînement et de test sont transformées en utilisant les composantes principales calculées précédemment.
- ❖ Entraînement du modèle SVM : Le modèle SVM est entraîné sur les données d'entraînement transformées par PCA.
- ❖ Prédictions et évaluation : Le modèle entraîné est utilisé pour faire des prédictions sur les données de test et l'exactitude du modèle est calculée à l'aide de `accuracy_score` et un rapport de classification est généré avec `classification_report`.
- ❖ Sauvegarde des modèles : Enfin, le modèle SVM entraîné est sauvegardé dans un fichier pickle pour une utilisation ultérieure sans avoir à re-entraîner le modèle à chaque fois.

```

app = Flask(__name__)
# Load the trained SVM model and PCA instance
svm_model = joblib.load('svm_model.pkl')
with open('pca_model.pkl', 'rb') as file:
    pca = pickle.load(file)
@app.route('/svm', methods=['POST'])
def predict_genre_svm():
    try:
        audio_file = request.files['audio_data']
        if audio_file is None:
            return jsonify({'error': 'No audio file received'})
        # Read the content of the audio file
        audio_content = audio_file.read()
        # Decode the audio data (if encoded, such as in base64)
        audio_bytes = base64.b64decode(audio_content) # Make sure to use the appropriate decoding method if necessary
        # Convert the audio data into a numpy array (floating-point)
        audio, sample_rate = librosa.load(io.BytesIO(audio_bytes), sr=None)
        # Example: Feature extraction using MFCC (adjust as needed)
        mfcc = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=58)

        # Apply PCA transformation using the previously fitted PCA instance with 21 components
        transformed_features = pca.transform(mfcc.T) # Transpose the MFCC features if needed

        # Make predictions with the trained SVM model
        predictions = svm_model.predict(transformed_features)
        # Map predicted genre labels to their names
        genre_names = {
            1: 'classical',
            2: 'blues',
            3: 'country',
            4: 'disco',
            5: 'electronic',
            6: 'folk',
            7: 'funk',
            8: 'jazz',
            9: 'pop',
            10: 'rock',
            11: 'soul',
            12: 'va'
        }
    except Exception as e:
        return jsonify({'error': str(e)})

```

Figure 5- Capture de service svm

4. Service vgg19

C'est un service qui permet de prédire le genre musical d'un fichier Wav passé en paramètre en suivant les étapes suivantes :

- Chargement de modèle
- Régler la route Endpoint Flask pour la prédiction avec svm : /svm
- Creation de spectrogram
- Exécuter la fonction `predict_genre_vgg19 ()`
- Exécution de l'application Flask : `If __name__ == '__main__': app.run (host='0.0.0.0', port=800)`

Pour la création de modèle vgg19 j'ai suivi dans mon code les étapes suivantes et en utilisant Kaggle Notebook en exécutant le fichier `constructionModelvgg19` dans mon projet a l'emplacement `nouvelle_arch\vgg19\constructionModelvgg19.py` :

- ❖ Chargement des exemples et Création d'un Spectrogramme : Le script charge un fichier audio d'exemple, en extrait un segment significatif, puis génère un spectrogramme à l'aide de la bibliothèque `librosa`.
- ❖ Préparation des Chemins d'Accès aux Images : Le script crée des dictionnaires (`train`, `val`, `test`) contenant des chemins d'accès aux images PNG correspondant à différentes partitions de l'ensemble de données de genres musicaux.
- ❖ Création du Dataset : À l'aide des chemins d'accès aux images, la fonction `createDataset ()` charge ces images, les redimensionne et les normalise, puis les prépare dans un format compatible avec TensorFlow pour l'entraînement du modèle.
- ❖ Prétraitement et Préparation des Données d'Entraînement, de Validation et de Test : Les données sont prétraitées et mises en forme pour l'entraînement, la validation et les tests à l'aide de TensorFlow.
- ❖ Définition et Entraînement du Modèle de Classification : Un modèle de transfert d'apprentissage utilisant l'architecture VGG19 pré-entraînée est construit et entraîné sur les données d'entraînement.
- ❖ Évaluation du Modèle : Une matrice de confusion est générée pour évaluer les performances du modèle sur les ensembles d'entraînement, de validation et de test.
- ❖ Sauvegarde du Modèle : Enfin, le modèle entraîné est sauvegardé au format `'vgg.h5'`.

```

@app.route('/predict', methods=['POST'])
def predict_genre():
    data = request.json
    image = data['image'] # Suppose you pass the image data in the request

    # Pré-traitement de l'image (adapter selon vos besoins)
    processed_image = preprocess_image(image)

    # Faire la prédiction avec le modèle
    predictions = model.predict(processed_image)
    predicted_genre = genreMap[np.argmax(predictions)]

    return jsonify({'predicted_genre': predicted_genre})

def preprocess_image(image):
    # Redimensionner l'image à la taille attendue par le modèle
    resized_image = image.resize((288, 432)) # Adapter à la taille attendue par votre modèle

    # Normaliser les pixels de l'image entre 0 et 1
    processed_image = np.array(resized_image) / 255.0

    # Adapter la forme pour correspondre à celle attendue par le modèle (si nécessaire)
    processed_image = np.expand_dims(processed_image, axis=0) # Si le modèle attend un lot (batch)

    return processed_image

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)

```

Figure 6- Capture de service vgg19

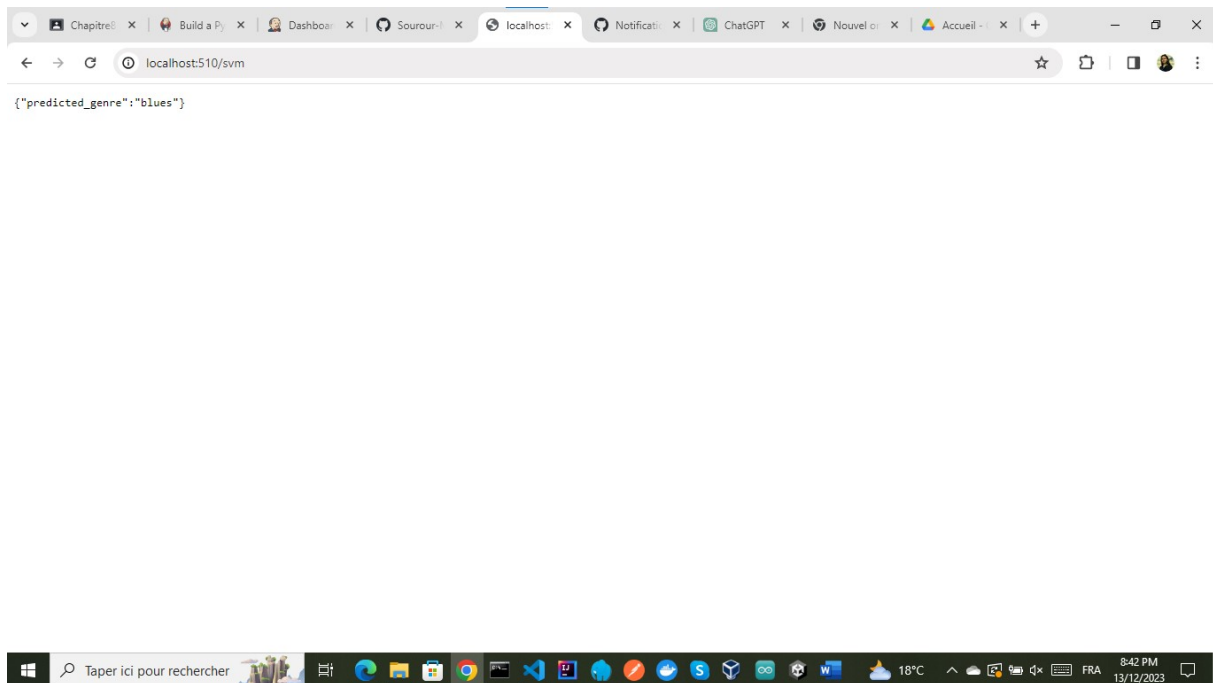
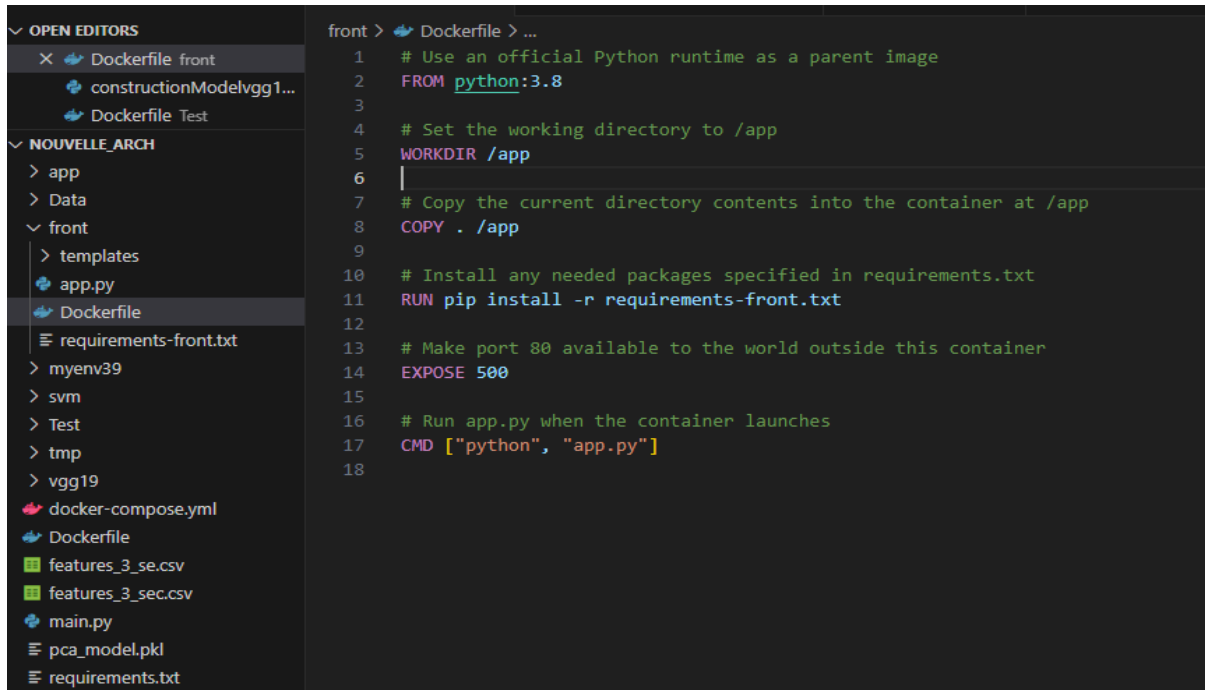


Figure 7- Prediction avec svm

5. Dockerfile pour le front-end

J'ai créé un Dockerfile pour la partie front-end de mon projet. Il contient un deux champs pour apporter le fichier Wav à prédire le genre et deux boutons l'un pour faire la prédiction avec le service svm et l'autre avec le service vgg19. Les Figure 8Figure 8 et

Figure 9 représentent respectivement le code et le résultat de front-end de mon application.



```
front > Dockerfile > ...
1  # Use an official Python runtime as a parent image
2  FROM python:3.8
3
4  # Set the working directory to /app
5  WORKDIR /app
6
7  # Copy the current directory contents into the container at /app
8  COPY . /app
9
10 # Install any needed packages specified in requirements.txt
11 RUN pip install -r requirements-front.txt
12
13 # Make port 80 available to the world outside this container
14 EXPOSE 500
15
16 # Run app.py when the container launches
17 CMD ["python", "app.py"]
18
```

Figure 8- Dockerfile de la partie front-end

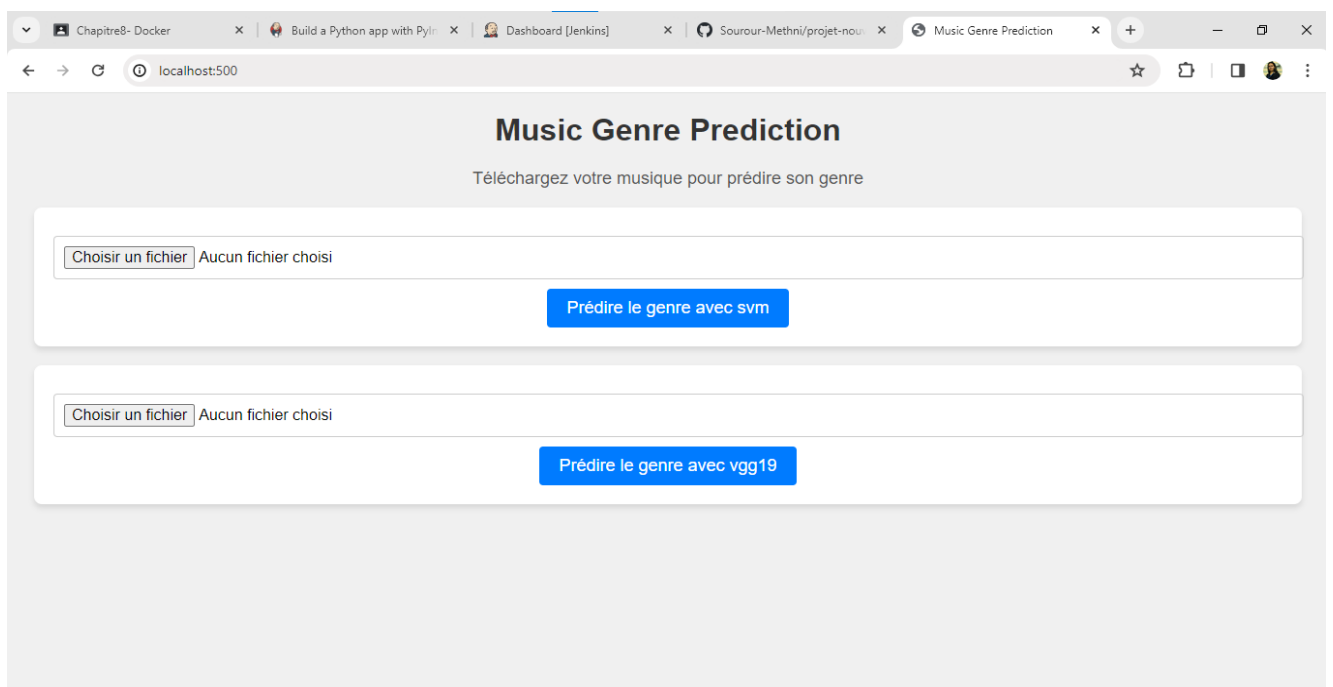
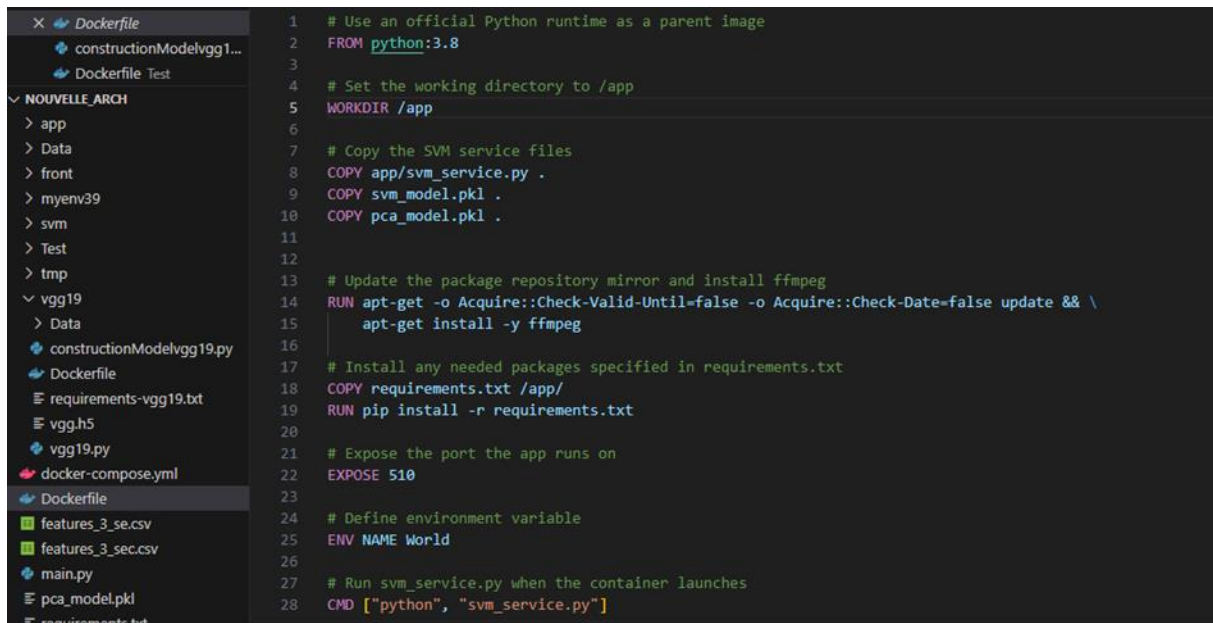


Figure 9- Front-end de mon projet

6. Dockerfile pour svm

Comme le montre la Figure 10, le conteneur de svm sera exécuté sur le port 510.

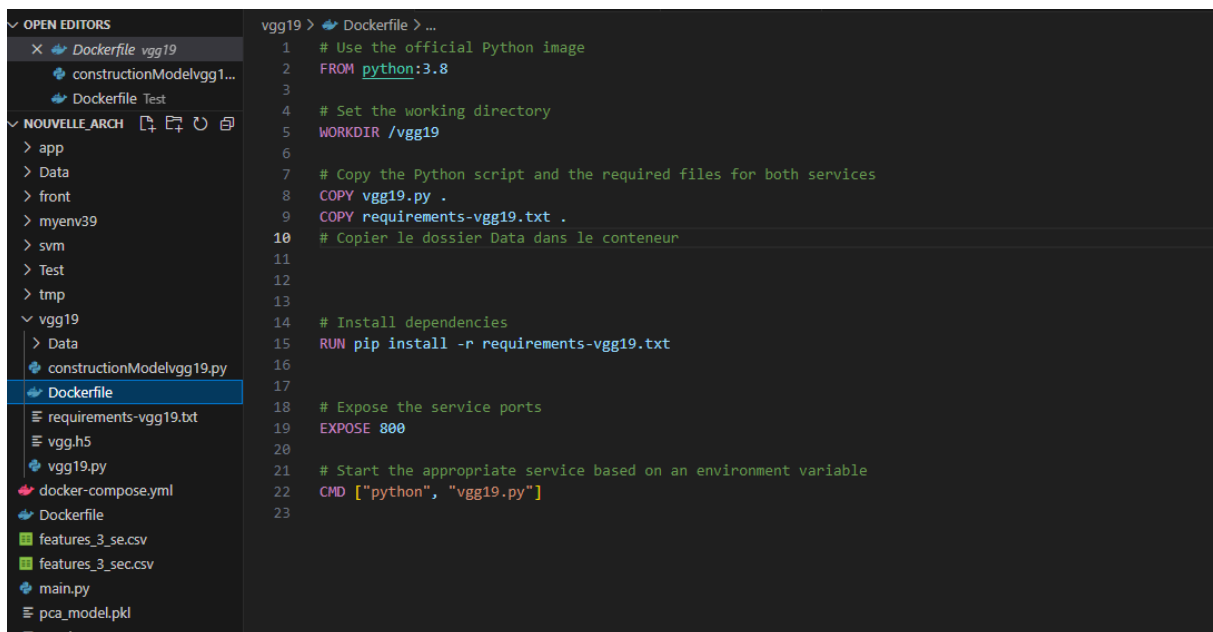


```
1 # Use an official Python runtime as a parent image
2 FROM python:3.8
3
4 # Set the working directory to /app
5 WORKDIR /app
6
7 # Copy the SVM service files
8 COPY app/svm_service.py .
9 COPY svm_model.pkl .
10 COPY pca_model.pkl .
11
12 # Update the package repository mirror and install ffmpeg
13 RUN apt-get -o Acquire::Check-Valid-Until=false -o Acquire::Check-Date=false update && \
14     apt-get install -y ffmpeg
15
16 # Install any needed packages specified in requirements.txt
17 COPY requirements.txt /app/
18 RUN pip install -r requirements.txt
19
20 # Expose the port the app runs on
21 EXPOSE 510
22
23 # Define environment variable
24 ENV NAME World
25
26 # Run svm_service.py when the container launches
27 CMD ["python", "svm_service.py"]
```

Figure 10- Dockerfile pour le service svm

7. Dockerfile pour vgg19

Comme le montre la Figure 11, le conteneur vgg19 sera exécuté sur le port 800.



```
1 # Use the official Python image
2 FROM python:3.8
3
4 # Set the working directory
5 WORKDIR /vgg19
6
7 # Copy the Python script and the required files for both services
8 COPY vgg19.py .
9 COPY requirements-vgg19.txt .
10 # Copier le dossier Data dans le conteneur
11
12 # Install dependencies
13 RUN pip install -r requirements-vgg19.txt
14
15 # Expose the service ports
16 EXPOSE 800
17
18 # Start the appropriate service based on an environment variable
19 CMD ["python", "vgg19.py"]
```

Figure 11- Dockerfile pour le service vgg19

8. Communication entre les trois conteneurs

La communication est assurée via un fichier docker-compose ou chaque conteneur (conteneur de front, conteneur de service svm, conteneur de service vgg19) sera lancé sur une adresse IP et port spécifique.

La commande : `docker-compose up --build` permet de lancer mon projet avec tous les conteneurs. Le résultat de l'exécution de cette commande est illustré par la Figure 12

Figure 12.

```
nouvelle_arch-app-service-1 | * Serving Flask app 'app' (lazy loading)
nouvelle_arch-app-service-1 | * Environment: production
nouvelle_arch-app-service-1 | WARNING: This is a development server. Do not use it in a production deployment.
nouvelle_arch-app-service-1 | Use a production WSGI server instead.
nouvelle_arch-app-service-1 | * Debug mode: off
nouvelle_arch-app-service-1 | * Running on all addresses.
nouvelle_arch-app-service-1 | WARNING: This is a development server. Do not use it in a production deployment.
nouvelle_arch-app-service-1 | * Running on http://192.168.16.3:500/ (Press CTRL+C to quit)
nouvelle_arch-app-service-1 | * Serving Flask app 'svm_service' (lazy loading)
nouvelle_arch-svm-service-1 | * Environment: production
nouvelle_arch-svm-service-1 | WARNING: This is a development server. Do not use it in a production deployment.
nouvelle_arch-svm-service-1 | Use a production WSGI server instead.
nouvelle_arch-svm-service-1 | * Debug mode: off
nouvelle_arch-svm-service-1 | /usr/local/lib/python3.8/site-packages/sklearn/base.py:348: InconsistentVersionWarning: Trying to unpickle estimator SVC from version 1.3.2 when using version 1.3.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
nouvelle_arch-svm-service-1 | https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
nouvelle_arch-svm-service-1 | warnings.warn(
nouvelle_arch-svm-service-1 | /usr/local/lib/python3.8/site-packages/sklearn/base.py:348: InconsistentVersionWarning: Trying to unpickle estimator PCA from version 1.3.2 when using version 1.3.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
nouvelle_arch-svm-service-1 | https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
nouvelle_arch-svm-service-1 | warnings.warn(
nouvelle_arch-svm-service-1 | * Running on all addresses.
nouvelle_arch-svm-service-1 | WARNING: This is a development server. Do not use it in a production deployment.
nouvelle_arch-svm-service-1 | * Running on http://192.168.16.2:510/ (Press CTRL+C to quit)
nouvelle_arch-svm-service-1 | 192.168.16.1 - - [13/Dec/2023 12:28:21] "GET / HTTP/1.1" 200 -
nouvelle_arch-app-service-1 |
```

Figure 12- Exécution de docker-compose

9. Tester avec Docker

Pour automatiser les tests j'ai créé un Jenkins file où il comportera un test pour la prédiction avec svm et test de nature de fichier.

Le contenu de Jenkins file est montré par la Figure 13.

```

Jenkinsfile U X test_flask_services.py
Jenkinsfile
1 pipeline {
2   agent any
3
4   stages {
5     stage('Clonage du code') {
6       steps {
7         git branch: 'main', url: 'https://github.com/Sourour-Methni/projet-nouvelle_arch.git'
8       }
9     }
10
11    stage('Installation des dépendances') {
12      steps {
13        sh 'pip install -r requirements.txt'
14      }
15    }
16
17    stage('Tests unitaires') {
18      steps {
19        sh 'python -m unittest tests/test_flask_services.py'
20      }
21      post {
22        always {
23          junit 'tests/results/unit_tests.xml'
24        }
25      }
26    }
27  }
28
29  post {
30    success {
31      echo 'Les tests ont réussi !'
32    }
33  }
34 }

```

Figure 13- Capture de Jenkinsfile

Le fichier test_flask_services.py pour les tests est présenté par la Figure 14.

```

import requests
import unittest

class TestIntegration(unittest.TestCase):

    def setUp(self):
        # Configurez votre environnement de tests pour les requêtes HTTP
        self.base_url = 'http://localhost:510'

    def test_svm_service_integration(self):
        data = {
            'wav_music': '/app/audio1_base64.txt'
        }
        response = requests.post(f'{self.base_url}/svm_service', json=data)
        self.assertEqual(response.status_code, 200)
        self.assertIn('genre', response.json()) # Vérifiez la présence de la clé 'genre' dans la réponse

    def test_predict_genre(self):
        data = {
            'wav_music': '/app/audio1_base64.txt'
        }
        response = self.app.post('/svm_service', json=data)
        self.assertEqual(response.status_code, 200)
        self.assertIn('genre', response.json())
        # Assurez-vous que la réponse est celle attendue
        expected_genre = 'rock' # Remplacez par le genre attendu pour vos données de test
        self.assertEqual(response.json()['genre'], expected_genre)

```

Figure 14- Fichier de tests

Conclusion

Ce mini-projet a démontré la mise en place d'un environnement complet pour la classification des genres musicaux à l'aide de modèles de Machine Learning déployés via des services web Flask dans des conteneurs Docker. L'utilisation de Docker a permis de gérer l'orchestration des services de manière efficace et portable.

L'intégration de Jenkins pour l'automatisation des processus a renforcé la reproductibilité et la robustesse du projet, facilitant le déploiement continu, les tests automatisés et les vérifications de code.

En conclusion, cette approche basée sur Docker, Flask et Jenkins a permis de créer un environnement cohérent et évolutif pour le déploiement et les tests d'applications basées sur Machine Learning, offrant une base solide pour de futurs développements et améliorations.