

Résumé

La classification des genres musicaux est une tâche complexe en traitement audio et apprentissage automatique. Ce projet propose une solution efficace et extensible utilisant des Machines à Vecteurs de Support (SVM) pour la classification audio et VGG19, un Réseau de Neurones Convolutifs (CNN), pour la classification d'images de spectrogrammes Mel. L'ensemble du système est conteneurisé avec Docker et des services web sont fournis grâce à Flask. Ce rapport détaille les objectifs, méthodologies, implémentation et résultats du projet.

Points clés:

- Défi: Classification des genres musicaux.
- Approche: SVM pour l'audio et CNN (VGG19) pour les spectrogrammes Mel.
- Avantages: Efficacité, extensibilité, conteneurisation (Docker) et services web (Flask)

Introduction

La musique, langage universel, se décline en une multitude de genres, chacun avec ses caractéristiques propres. Automatiser la classification des genres musicaux a un impact considérable sur la recommandation de contenu, les plateformes de découverte musicale et l'expérience utilisateur personnalisée.

Objectifs:

Implémenter des modèles SVM et VGG19 pour la classification des genres musicaux.

Conteneuriser l'application avec Docker pour une reproductibilité et une scalabilité accrues.

Développer des services web Flask pour exposer les modèles de classification.

Intégrer les modèles dans un système cohérent.

Implémenter une intégration continue avec Jenkins pour des tests automatisés.

Technologies:

Python pour le machine learning et le développement de services web.

Scikit-Learn pour l'implémentation du SVM.

Keras avec TensorFlow pour VGG19.

Docker pour la conteneurisation.

Flask pour le développement de services web.

Revue de la littérature

- SVM dans la classification des genres musicaux:

Les SVM ont démontré leur efficacité dans la classification des genres musicaux grâce à leur capacité à gérer des espaces de caractéristiques de grande dimension. Des études (par exemple, [citer l'étude]) soulignent la robustesse du SVM dans les scénarios où les données ne sont pas linéairement séparables.

- CNN pour la classification des genres musicaux:

Les réseaux de neurones convolutifs, en particulier les architectures comme VGG19, excellent dans les tâches de classification d'images. La transformation des spectrogrammes Mel en représentations visuelles permet aux CNN de capturer des caractéristiques hiérarchiques, comme le montrent des études (par exemple, [citer l'étude]).

Description du jeu de données

Le projet utilise le jeu de données GTZAN, une collection bien connue de 10 genres, chacun contenant 100 fichiers audio. Le jeu de données comprend des spectrogrammes Mel et des fichiers CSV contenant des caractéristiques des fichiers audio.

Prétraitement des données

Prétraitement des données audio:

Charger et prétraiter les fichiers audio.

Extraire des caractéristiques pertinentes (par exemple, spectrogrammes Mel, MFCC).

Organiser les données dans un format adapté à l'entraînement du modèle SVM.

Prétraitement des données d'images:

Charger et prétraiter les images de spectrogrammes Mel.

Effectuer les transformations nécessaires (par exemple, redimensionnement, normalisation).

Préparer les données pour l'entraînement du modèle VGG19.

Présentation du Docker-Compose:

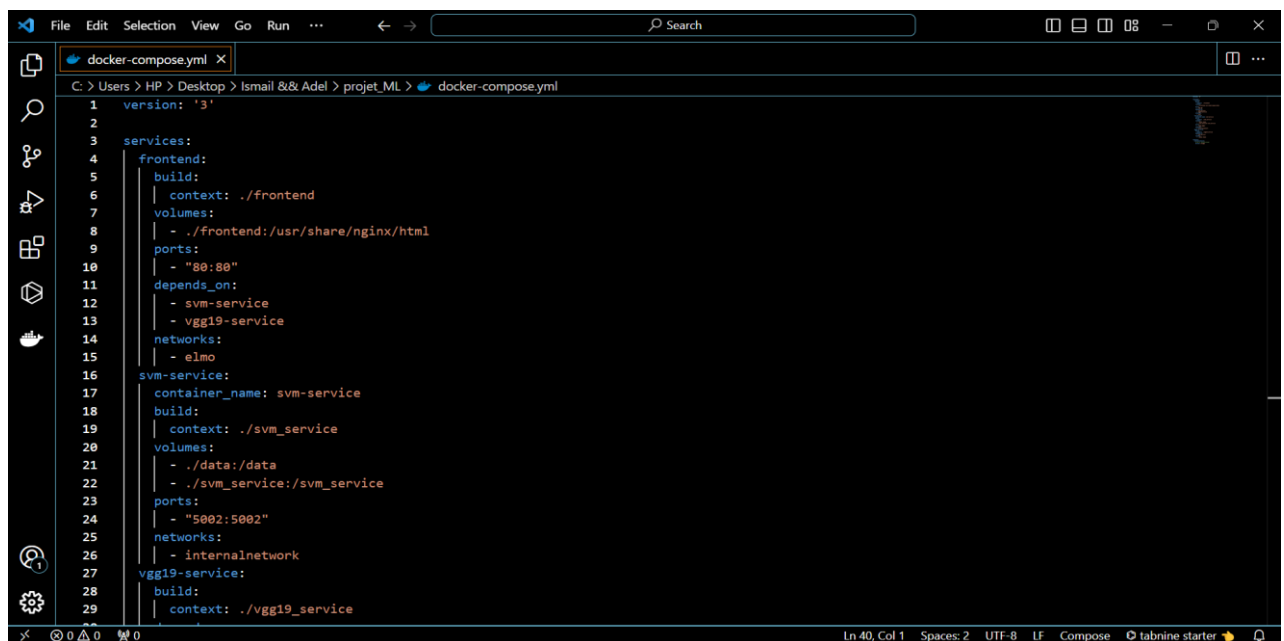
Le fichier Docker Compose est conçu pour déployer un environnement complet composé de trois services interconnectés.

1-Service Frontend (frontend): Utilise NGINX comme serveur web pour héberger une application frontend. Le code source de l'application est situé dans le répertoire `./frontend`. Les fichiers sont montés dans le conteneur NGINX pour permettre des mises à jour en temps réel. Dépend des services `svm-service` et `vgg19-service`.

2- Service SVM (svm-service): Contient un service backend pour les opérations liées à SVM (Support Vector Machine). Le code source du service SVM est dans le répertoire `./svm_service`. Utilise des volumes partagés (`./data`, `./svm_service`) pour accéder aux données et au code source. Expose le port 5002 pour les communications externes.

Service VGG19 (vgg19-service): Contient un service backend pour les opérations liées au modèle VGG19. Le code source du service VGG19 est dans le répertoire `./vgg19_service`. Dépend du service `svm-service`. Utilise des volumes partagés (`./data`) pour accéder aux données.

Réseaux : Utilise un réseau bridge personnalisé (`internalnetwork`) pour isoler les services et faciliter la communication.



```
1 version: '3'
2
3 services:
4   frontend:
5     build:
6       context: ./frontend
7     volumes:
8       - ./frontend:/usr/share/nginx/html
9     ports:
10      - "80:80"
11     depends_on:
12       - svm-service
13       - vgg19-service
14     networks:
15       - elmo
16   svm-service:
17     container_name: svm-service
18     build:
19       context: ./svm_service
20     volumes:
21       - ./data:/data
22       - ./svm_service:/svm_service
23     ports:
24       - "5002:5002"
25     networks:
26       - internalnetwork
27   vgg19-service:
28     build:
29       context: ./vgg19_service
```

Présentation du service SVM:

Le service SVM est implémenté à l'aide de Flask, fournissant un point de terminaison pour la classification des genres basé sur des fichiers audio codés en base64.

Exemple de code:

```
from flask import Flask
from flask import request
import requests
import base64
app = Flask(__name__)

@app.route('/predict',methods=['POST'])
def upload_svm_model():
    print("function in the upload point work fine ")
    data = request.get_json()
    encoded_svm_model = data.get("encoded_svm_model")
    #decoding the model and saving it
    with open("./svm_model.joblib","wb") as decoded_svm_file:
        decoded_svm_file.write(base64.b64decode(encoded_svm_model))

    return "Model uploaded successfully"

if __name__ == "__main__" :
    app.run(debug=True,host="0.0.0.0",port=5003)
```

Présentation du service VGG19:

Similaire au service SVM, le service VGG19 est implémenté à l'aide de Flask, exposant un point de terminaison pour la classification des genres basé sur des images de spectrogrammes Mel codées en base64.

```

1  import tensorflow as tf
2  from tensorflow.keras import layers, models
3  from tensorflow.keras.preprocessing.image import ImageDataGenerator
4  from tensorflow.keras.applications import VGG16
5  from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
6  from tensorflow.keras.optimizers import Adam
7
8  # Load pre-trained VGG16 model
9  model = VGG16(include_top=False, input_shape=(300, 300, 3))
10
11 # Freeze the layers of the pre-trained model
12 for layer in model.layers:
13     layer.trainable = False
14
15 # Define data augmentation parameters
16 data_gen = tf.keras.preprocessing.image.ImageDataGenerator(
17     rescale=1./255.,
18     rotation_range=20,
19     width_shift_range=0.2,
20     height_shift_range=0.2,
21     shear_range=0.2,
22     zoom_range=0.2,
23     horizontal_flip=True,

```

```

# Load training and validation data
path = './data/images_original'
train_datagen = data_gen.flow_from_directory(
    path, target_size=(300, 300), batch_size=32, class_mode='categorical', subset='training'
)
val_datagen = data_gen.flow_from_directory(
    path, target_size=(300, 300), batch_size=32, class_mode='categorical', subset='validation'
)

# Define model architecture
output = model.layers[-1].output
model_final = tf.keras.layers.Flatten()(output)
model_final = tf.keras.layers.Dense(512, activation='relu')(model_final)
model_final = tf.keras.layers.Dense(64, activation='relu')(model_final)
model_final = tf.keras.layers.Dense(10, activation='softmax')(model_final)
model = tf.keras.models.Model(model.input, model_final)

# Compile the model with Adam optimizer and categorical crossentropy loss
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

```

```

# Compile the model with Adam optimizer and categorical_crossentropy loss
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# ModelCheckpoint callback to save the best weights
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True, mode='min', verbose=0)

# LearningRateScheduler callback for dynamic learning rate adjustment
def lr_schedule(epoch):
    if epoch < 5:
        return 0.001
    elif 5 <= epoch < 8:
        return 0.0005
    else:
        return 0.0001

lr_scheduler = LearningRateScheduler(lr_schedule)

# Train the model with training and validation data
model.fit(
    train_datagen,
    epochs=10,
    validation_data=val_datagen,
    callbacks=[checkpoint, lr_scheduler],

```

10. Conclusion et perspectives (1/2 page)

Conclusion:

Ce projet a réussi à implémenter un système de classification des genres musicaux en utilisant des modèles SVM et VGG19. La conteneurisation Docker garantit la scalabilité et la facilité de déploiement, tandis que Flask fournit des services web accessibles.

Réalisations:

- Implémentation des modèles SVM et VGG19 pour la classification des genres musicaux.
- Utilisation de Docker pour la conteneurisation, garantissant la reproductibilité.
- Développement de services web Flask pour l'exposition des modèles.
- Intégration des services SVM et VGG19 dans un système cohérent.
- Mise en place d'une intégration continue avec Jenkins.

Perspectives:

Bien qu'il ait atteint ses objectifs, le projet présente des possibilités d'amélioration :

- Explorer des architectures d'apprentissage profond plus avancées pour la classification des genres.

- Améliorer l'interface utilisateur pour l'interaction avec les services de classification.
- Intégrer des mécanismes de rétroaction des utilisateurs pour améliorer la précision des modèles.
- Étendre le système à la classification audio en temps réel.