

***Rapport du Projet***  
***Nouvelles Architectures***

**Réalisé par :**

**YAAKOUBI Siwar 3émeNTS**

**JAFFELI Sarra 3émeNTS**

# Sommaire

Chapitre 1 : Etude Théorique .....	4
1. Docker .....	4
2. Flask .....	4
3. Docker-compose.....	4
4. SVM.....	4
5. VGG.....	4
Chapitre 2 : Etude Technique.....	5
1. Environnement matériel .....	5
2. Environnement logiciel .....	5
Chapitre 3 : Réalisation .....	7
1. Architecture.....	7
2. Structure du projet.....	7
3. Scénario d'exécution .....	12
3.1. Page d'Accueil .....	12

## Table des Figures

Figure 1 docker-compose.yml.....	7
Figure 2 La structure du projet.....	8
Figure 3 Code Dockerfile.....	8
Figure 4Répertoire Uploads .....	9
Figure 5 Requirements.txt .....	9
Figure 6 App.py.....	10
Figure 7 Dockerfile .....	10
Figure 8 Partie du code html .....	11
Figure 9App.py.....	11
Figure 10 Requirements.txt .....	12
Figure 11 page d'accueil.....	12
Figure 12 choix d'un fichier audio.....	13
Figure 13 Le résultat SVM.....	13
Figure 14 choix d'une image .....	14
Figure 15 Le résultat VGG.....	14

# **Chapitre 1 : Etude Théorique**

## **1. Docker**

Docker représente une technologie en code source libre qui donne aux utilisateurs la capacité de concevoir, administrer et lancer des applications à travers l'utilisation de conteneurs. Les applications qui s'exécutent sur Docker sont encapsulées dans leur propre environnement, où elles portent les ressources nécessaires à leur fonctionnement.

## **2. Flask**

Flask, écrit en Python, se présente comme un framework pour le développement d'applications web. Il propose divers modules qui simplifient la tâche des développeurs en leur permettant de créer des applications sans se préoccuper des aspects détaillés tels que la gestion des protocoles, des threads, et autres.

## **3. Docker-compose**

Docker Compose constitue un outil conçu pour définir et exécuter des applications Docker composées de plusieurs conteneurs. Avec Compose, la configuration des services de votre application s'effectue à l'aide d'un fichier YAML. Ensuite, vous pouvez créer et démarrer tous les services selon cette configuration en utilisant une seule commande.

## **4. SVM**

Les SVMs, ou Machines à Vecteurs de Support, constituent une catégorie d'algorithmes d'apprentissage automatique qui peuvent résoudre une variété de problèmes, qu'il s'agisse de classification, de régression ou de détection d'anomalies. Ceux-ci sont réputés pour leurs garanties théoriques solides, leur grande adaptabilité et leur facilité d'utilisation, même pour des utilisateurs ne possédant pas une expertise approfondie en data mining.

## **5. VGG**

VGG est un réseau de neurones convolutionnels qui a acquis une renommée en remportant la compétition ILSVRC (ImageNet Large Scale Visual Recognition Challenge) en 2014. Ce modèle a atteint une impressionnante précision de 92,7% sur ImageNet, établissant l'un des meilleurs scores à ce jour. Sa particularité réside dans l'optimisation des couches de convolution, introduisant des noyaux de convolution de dimensions réduites (3×3), marquant ainsi une avancée par rapport aux modèles antérieurs. Son entraînement sur plusieurs semaines a nécessité l'utilisation de cartes graphiques de pointe.

# Chapitre 2 : Etude Technique

## 1. Environnement matériel

Pour la réalisation de ce projet j'ai utilisé le matériel décrit par le tableau suivant :

Système d'exploitation	Windows 11
Processeur	Intel Core AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz
Nombre de cœurs	4 cœurs
Mémoire RAM	20,0 Go
Disque dur	512 Go

## 2. Environnement logiciel

- **Visual Studio Code**

Visual Studio Code, créé par Microsoft, est un éditeur de code extensible compatible avec Windows, Linux et macOS. Ses caractéristiques englobent le débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, la réfactorisation du code, ainsi que l'intégration de Git. Les utilisateurs ont la possibilité de personnaliser le thème, les préférences et d'installer des extensions ajoutant des fonctionnalités supplémentaires. Le code source de Visual Studio Code est issu d'un projet logiciel libre et open source.

- **Docker**

Docker représente une plateforme permettant d'exécuter des applications spécifiques au sein de conteneurs logiciels. Selon les analyses de la firme de recherche industrielle 451 Research, Docker est défini comme un outil capable de regrouper une application ainsi que ses dépendances au sein d'un conteneur isolé, ce dernier pouvant être exécuté sur n'importe quel serveur.

- **Flask**

Flask est un micro-framework open-source de développement web en Python. Il se qualifie en tant que micro-framework en raison de sa légèreté. L'objectif de Flask est de maintenir un noyau simple tout en offrant une extensibilité.

- **TensorFlow**

TensorFlow est un outil open source d'apprentissage automatique développé par Google. Le code source a été rendu public le 9 novembre 2015 par Google et est disponible sous la licence Apache. Il repose sur l'infrastructure DistBelief, initiée par

Google en 2011, et offre une interface pour les langages de programmation Python, Julia et R.

- **Librosa**

Librosa est une bibliothèque d'analyse de la musique et des fichiers audio.

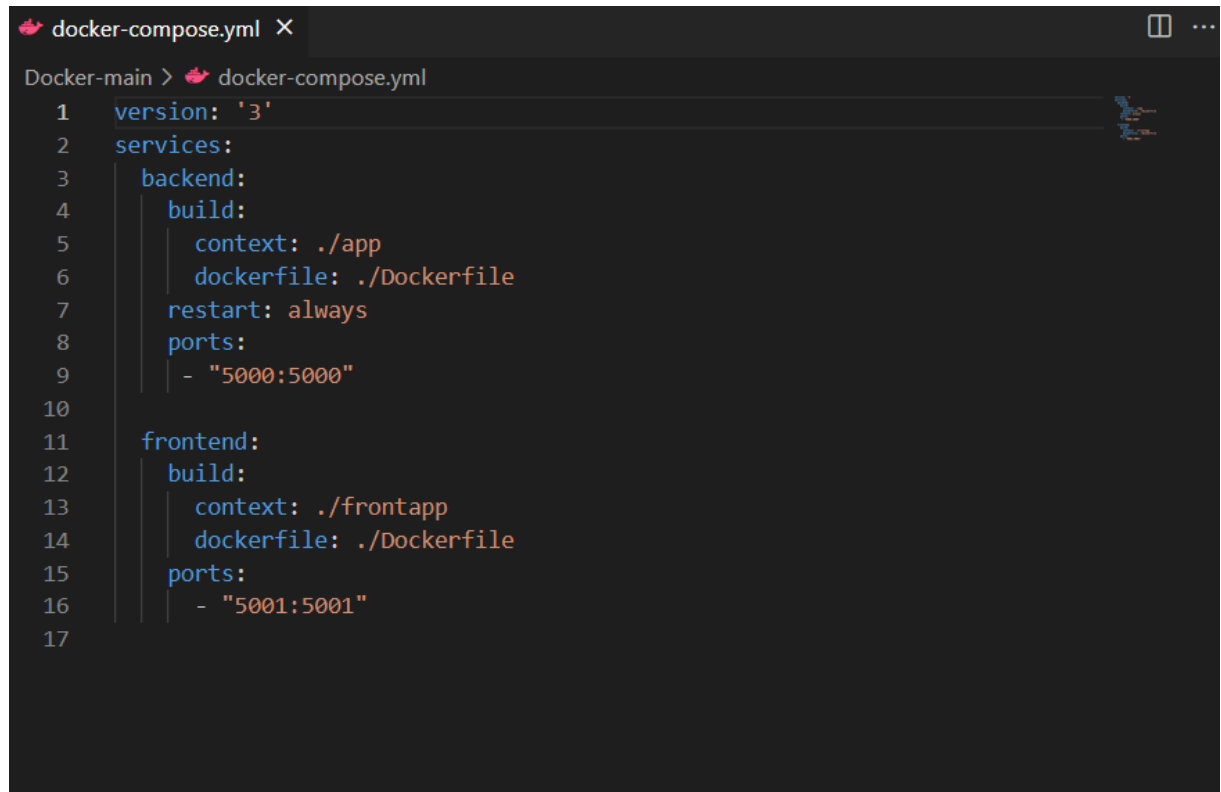
# Chapitre 3 : Réalisation

## 1. Architecture

Nous avons deux containers :

- Un pour la partie Backend
- L'autre pour la partie Front

Ces deux containers sont orchestrés par Docker-compose



```
1 version: '3'
2 services:
3   backend:
4     build:
5       context: ./app
6       dockerfile: ./Dockerfile
7     restart: always
8     ports:
9       - "5000:5000"
10
11   frontend:
12     build:
13       context: ./frontapp
14       dockerfile: ./Dockerfile
15     ports:
16       - "5001:5001"
17
```

Figure 1 docker-compose.yml

## 2. Structure du projet

Le projet se constitue de deux grandes parties comme illustre la figure suivante :

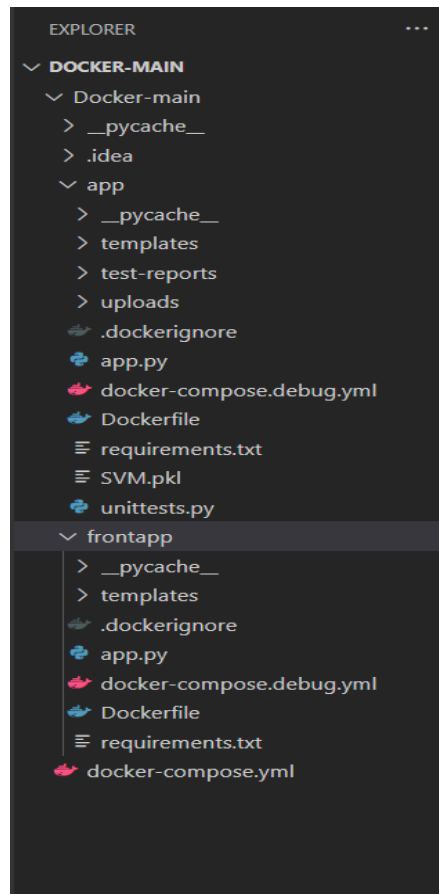


Figure 2 La structure du projet

- **App** : C'est un projet Flask dédié au Backend composé de :
  - Dockerfile : voici le code généré dans Dockerfile :

```

Dockerfile X
Docker-main > app > Dockerfile
1  # For more information, please refer to https://aka.ms/vscode-docker-python
2  FROM python:3.7-slim-buster
3
4  EXPOSE 5000
5
6  # Keeps Python from generating .pyc files in the container
7  ENV PYTHONDONTWRITEBYTECODE=1
8
9  # Turns off buffering for easier container logging
10 ENV PYTHONUNBUFFERED=1
11
12 # Install pip requirements
13 COPY requirements.txt .
14 RUN python -m pip install -r requirements.txt
15 RUN pip install soundfile
16 WORKDIR /app
17 COPY . /app
18 ENV NUMBA_CACHE_DIR=/tmp/numba_cache
19
20
21 # Switching to a non-root user, please refer to https://aka.ms/vscode-docker-python-user-rights
22 RUN useradd appuser && chown -R appuser /app
23 USER appuser
24
25 # During debugging, this entry point will be overridden. For more information, please refer to https://aka.ms/vscode-docker-python-debug
26 CMD ["python", "app.py"]
27

```

Figure 3 Code Dockerfile



- Uploads : Le répertoire où les fichiers sont chargés.

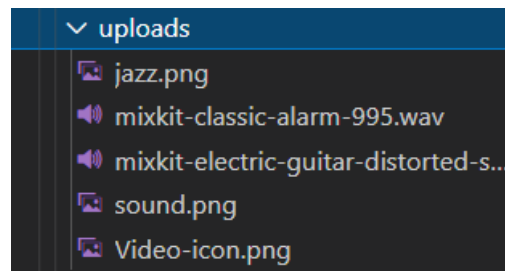


Figure 4 Répertoire Uploads

- Requirements.txt : Une liste des différentes bibliothèques nécessaires.

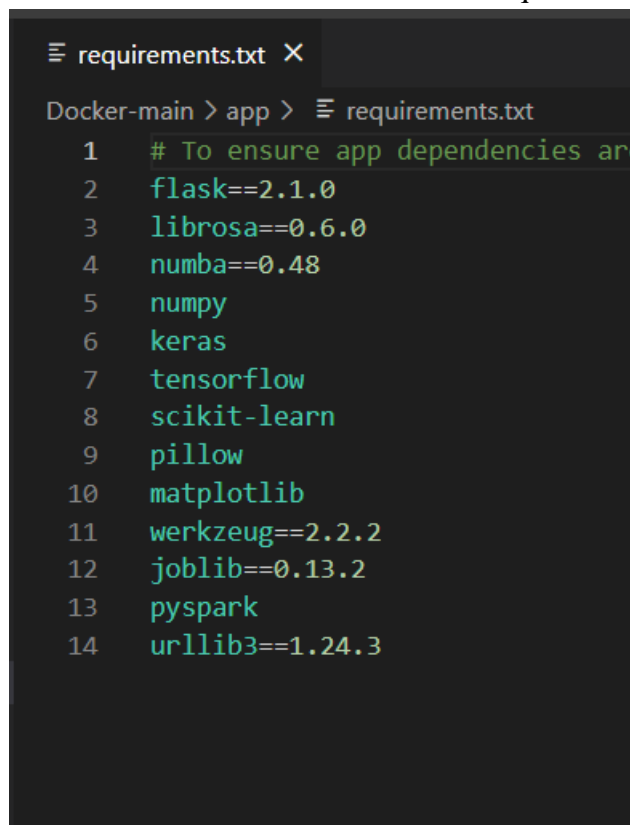


Figure 5 Requirements.txt

- App.py : Un fichier Python qui comprend les diverses importations ainsi que les contrôleurs utilisés pour chaque requête HTTP.

```
appp.py x frontapp.py requirements.txt app\Dockfile frontapp\Dockfile
1 from flask import Flask
2 from flask import render_template, request
3 app=Flask(__name__, template_folder='templates')
4 app=Flask(__name__)
5
6 from werkzeug.utils import secure_filename
7
8 import os
9 from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
10 from PIL import Image
11 from keras.applications.vgg19 import VGG19
12 from keras.preprocessing import image
13 from keras.applications.vgg19 import preprocess_input
14 from keras.models import Model
15 import numpy as np
16 from tensorflow.keras.preprocessing.image import load_img
17 import librosa.display
18 import scipy.io.wavfile as wavfile
19 import numpy
20 import os.path
21 from os import walk
```

Figure 6 App.py

- **FrontApp** : Il s'agit d'un projet Flask dédié au frontend, comprenant :
- Dockerfile

```
Dockerfile X
Docker-main > frontapp > Dockerfile
1 # For more information, please refer to https://aka.ms/vscode-docker-python
2 FROM python:3.7-slim-buster
3
4 EXPOSE 5001
5
6 # Keeps Python from generating .pyc files in the container
7 ENV PYTHONDONTWRITEBYTECODE=1
8
9 # Turns off buffering for easier container logging
10 ENV PYTHONUNBUFFERED=1
11
12 # Install pip requirements
13 COPY requirements.txt .
14 RUN python -m pip install -r requirements.txt
15
16 WORKDIR /app
17 COPY . /app
18
19 # Switching to a non-root user, please refer to https://aka.ms/vscode-docker-python-user-rights
20 RUN useradd appuser && chown -R appuser /app
21 USER appuser
22
23 # During debugging, this entry point will be overridden. For more information, please refer to https://aka.ms/vscode-docker-python
24 CMD ["python", "app.py"]
```

Figure 7 Dockerfile

- **Templates** : Ce dossier contient les fichiers HTML du projet.

```

<!DOCTYPE html>
<html>
<head>
<style>
  body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin: 20px;
  }

  h1 {
    color: #333;
  }

  form {
    margin-top: 20px;
  }

  input[type="file"] {
    display: none; /* Hide the default file input */
  }

  .custom-file-upload {
    border: 1px solid #ccc;
    display: inline-block;
    padding: 6px 12px;
    cursor: pointer;
    background-color: #4CAF50;
    color: white;
    border-radius: 5px;
  }

  .custom-file-upload:hover {
    background-color: #45a049;
  }

  input[type="submit"] {
    background-color: #4CAF50;
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
  }

  input[type="submit"]:hover {
    background-color: #45a049;
  }
</style>
</head>
<body>
<h1>VGG</h1>
<form action="http://localhost:5000/uploadervgg" method="POST" enctype="multipart/form-data">
  <label for="vgg-file" class="custom-file-upload">
    Select File
  </label>
  <input type="file" name="file" id="vgg-file" />
  <input type="submit" />
</form>
</body>
</html>

```

Figure 8 Partie du code html

- App.py : Un fichier Python responsable du "rendering" des templates.

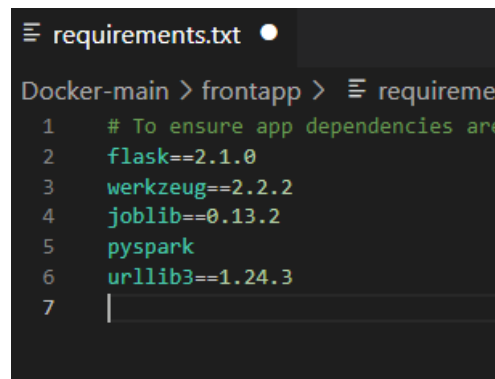
```

app\app.py  frontapp\app.py  requirements.txt
1  from flask import Flask
2  from flask import render_template
3  app=Flask(__name__, template_folder='templates')
4
5
6  @app.route('/home')
7  def upload_files():
8      return render_template("form.html")
9
10
11  if __name__ == "__main__":
12      app.run(host="0.0.0.0", port=5001)

```

Figure 9 App.py

- Requirements.txt : Une liste des différentes bibliothèques nécessaires.



```
requirements.txt
Docker-main > frontapp > requirements.txt
1 # To ensure app dependencies are
2 flask==2.1.0
3 werkzeug==2.2.2
4 joblib==0.13.2
5 pyspark
6 urllib3==1.24.3
7
```

Figure 10 Requirements.txt

### 3. Scénario d'exécution

#### 3.1. Page d'Accueil

##### 3.1.1 SVM

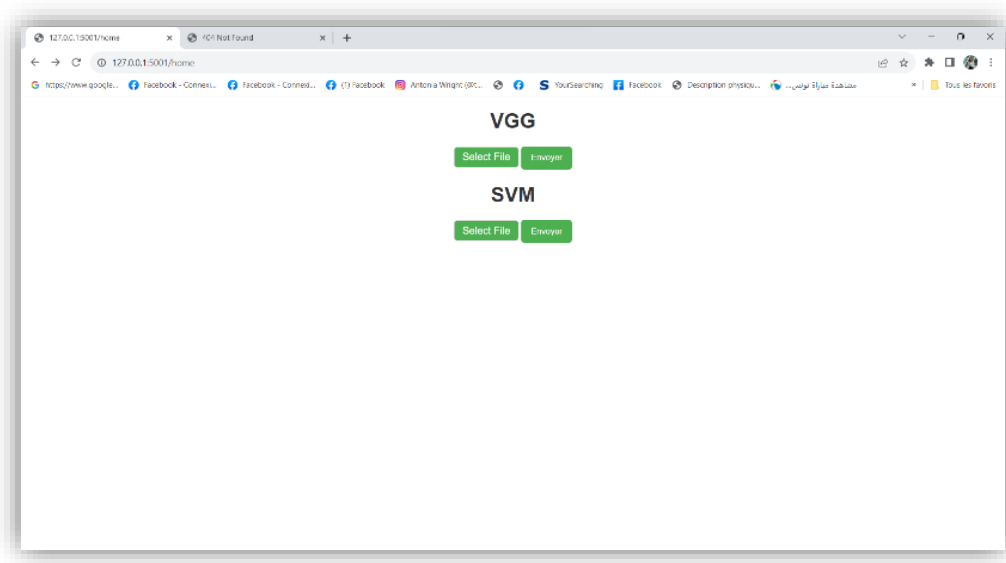
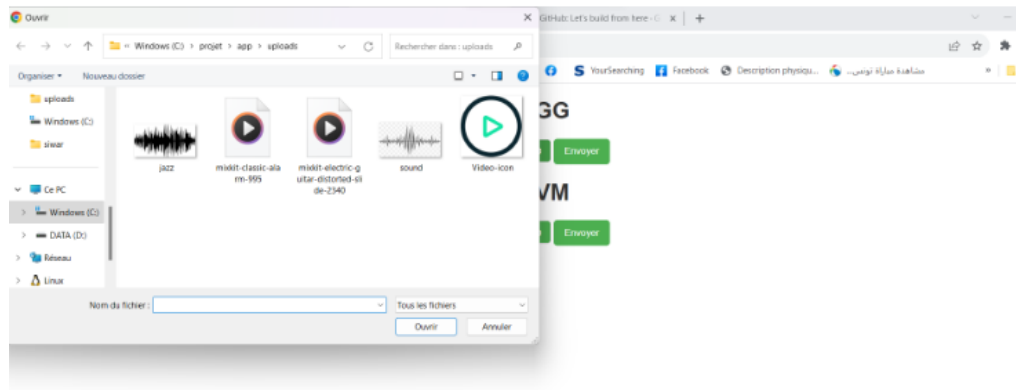


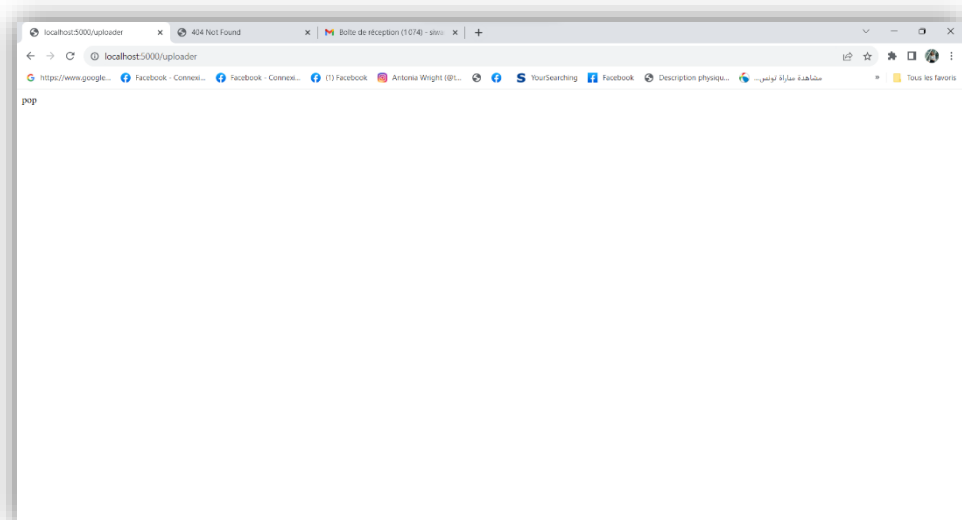
Figure 11 page d'accueil

Une fois nous avons accéder à la page d'accueil, nous pouvons maintenant choisir un fichier audio de type wav pour le déposer.



**Figure 12** choix d'un fichier audio

On peut maintenant voir le résultat en cliquant sur soumettre

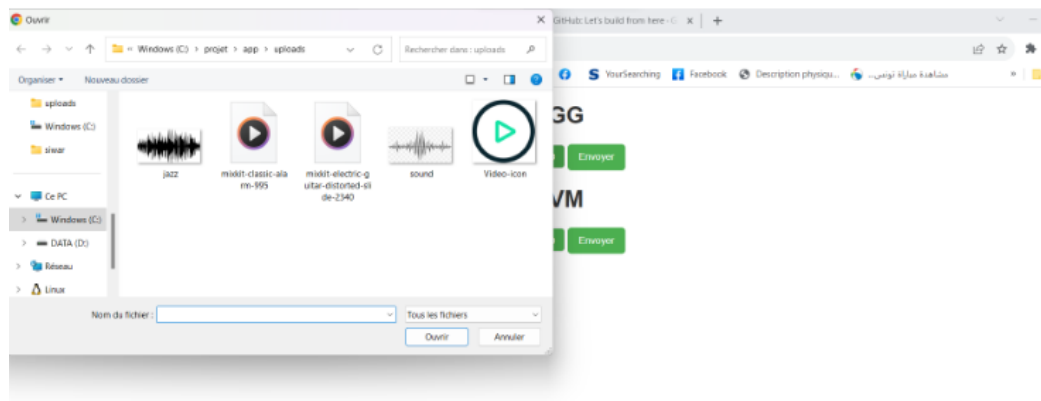


**Figure 13** Le résultat SVM

### 3.2.1 VGG

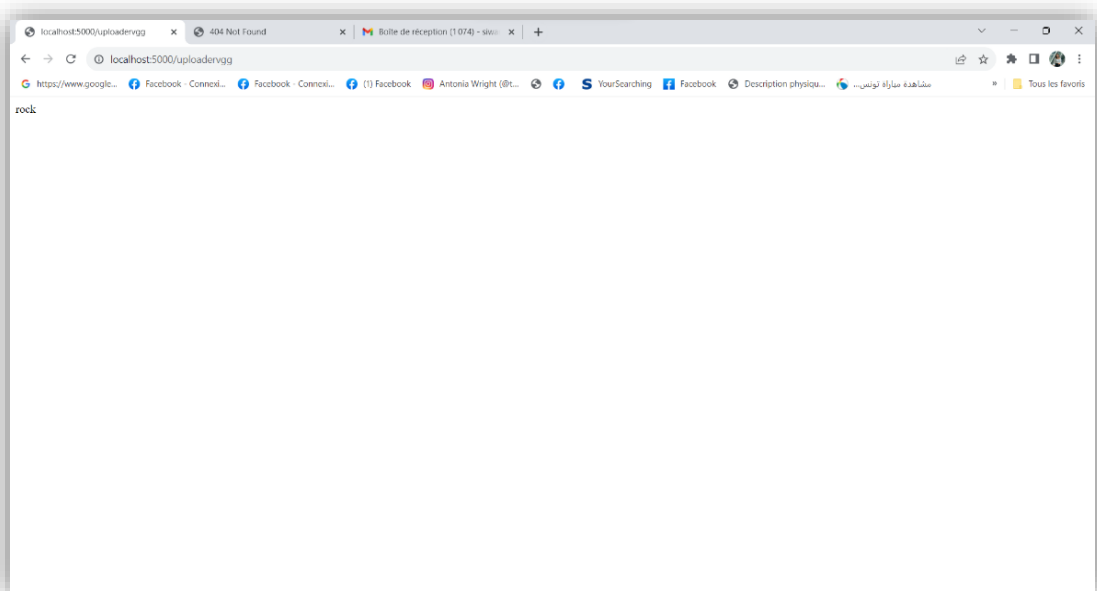
De même pour les vgg c'est la même page d'accueil qui s'affiche

Nous devons choisir une image de spectrogramme sous le titre VGG



**Figure 14** choix d'une image

Puis le résultat va être affiché comme montre la figure



**Figure 15** Le résultat VGG