Détail sur l'implémentation du projet « Poubelle Intelligente »

Les données utilisées pour l'entraînement du modèle proviennent de **Kaggle**. Le jeu de données contient des images classées en **six catégories distinctes**.

Les images du dataset sont réparties en **six classes** représentant différents types de déchets recyclables :

- **Cardboard** : Carton et emballages en papier épais.
- **Glass** : Bouteilles, bocaux et autres objets en verre.
- **Metal** : Canettes, boîtes de conserve et autres objets métalliques.
- **Paper** : Journaux, magazines, feuilles et emballages en papier.
- **Plastic** : Bouteilles en plastique, sacs et autres objets plastiques.
- **Trash** : Déchets non recyclables, mélanges de matières ou objets non catégorisés.

Les données ont été téléchargées depuis un dataset existant sur Internet.

Le modèle utilisé est **MobileNetV2**, un **réseau de neurones convolutionnel** léger et efficace développé avec **TensorFlow/Keras**. Il est conçu pour classifier des images en **six catégories différentes**. L'entraînement est réalisé dans le notebook **entrainement_model.ipynb**.

- **Modèle de base** : MobileNetV2 pré-entraîné sur **ImageNet**, sans la couche de classification finale.
- **Couches additionnelles** :
  - GlobalAveragePooling2D() : Réduction de la dimension des caractéristiques.
  - Dense(128, activation="relu") : Couche dense avec activation **ReLU**.
  - Dropout(0.5) : Réduction du surapprentissage.
  - Dense(6, activation="softmax") : Classification finale en **6 catégories**.

- **Redimensionnement** des images à 224x224 pixels.

- **Normalisation** des pixels (valeurs entre 0 et 1).
- **Augmentation des données** *(rotation, zoom, flip)* pour améliorer la **généralisation**.

- Chargement de **MobileNetV2** sans sa couche de classification (include_top=False).
- Gel des couches du modèle pré-entraîné pour conserver les **poids de base**.
- Ajout des **couches personnalisées** mentionnées ci-dessus.

- **Optimiseur** : Adam avec un **learning rate** ajusté.
- **Fonction de perte** : Categorical Crossentropy.
- **Métriques suivies** : Accuracy.
- **Entraînement** sur un **jeu d'entraînement** avec **validation** sur un **jeu de test**.

Le modèle est évalué sur le **dataset de validation**.

- **Perte (val_loss)** et **Précision (val_accuracy)** sont calculées.
- **Affichage des résultats** après l'entraînement pour évaluer la **performance globale**.

- Enregistrement sous format **.h5** (model.h5) pour une **réutilisation** et un **déploiement** facile.

L'API a été développée avec **FastAPI** pour gérer la prédiction du modèle de reconnaissance d'objets recyclables. Elle permet d'envoyer une image et de recevoir une estimation de la recyclabilité de l'objet.

Fonctionnalités de l'API :

- Un endpoint **GET /** qui renvoie un message de bienvenue.

- Un endpoint **POST** /**predict** qui :

1. Reçoit une image en tant que fichier uploadé.

2. Effectue un prétraitement de l’image (redimensionnement, conversion en tableau

numpy, ajout d’une dimension).

3. Utilise un modèle de classification entraîné pour prédire si l’objet est recyclable ou

non.

4. Retourne la probabilité associée à la recyclabilité de l’objet.

L’interface utilisateur a été conçue avec **Streamlit** pour permettre une interaction simple et intuitive

avec le modèle.

Fonctionnalités de l’interface :

- Un champ permettant à l’utilisateur de télécharger une image.

- Une visualisation de l’image uploadée.

- Une requête envoyée à l’API pour obtenir une prédiction en temps réel.

- Un affichage des résultats avec un message de confirmation sur le recyclage de l’objet :

. Si la probabilité de recyclabilité est supérieure à 50%, l’objet est considéré comme recyclable.

. Sinon, il est classé comme non recyclable.

Utilisation de l’application Streamlit avec l’api en local :

- Installation des dépendances : **pip install -r requirements.txt**

- Démarrer l’API : **uvicorn api:app –reload**

- Lancer l’interface utilisateur Streamlit : **streamlit run frontend.py**

- Ouvrir l’URL locale fournie par Streamlit

- Charger une image d’objet

- Obtenir la classification de l’objet en temps réel

Ce guide explique comment dockeriser et déployer votre projet avec Docker et FastAPI.

Assurez-vous d'avoir Docker installé sur votre machine. Vous pouvez l'installer en suivant les instructions officielles : Docker Installation.

Vérifiez l'installation en exécutant la commande :

```
docker --version
```

Le fichier requirements.txt contient les dépendances nécessaires au projet. Créez-le et ajoutez les dépendances suivantes :

```
fastapi==0.103.1
uvicorn
python-multipart
tensorflow
pillow
```

Créez un fichier nommé Dockerfile (sans extension) et ajoutez-y le contenu suivant :

```
# Utilisation d'une image Python allégée
FROM python:3.10-slim

# Définition du répertoire de travail
WORKDIR /app

# Copie des dépendances
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copie des fichiers du projet
COPY . .

# Exposition du port 8000
EXPOSE 8000

# Commande pour démarrer l'application
CMD ["uvicorn", "api:app", "--host", "0.0.0.0", "--port", "8000"]
```

4. Construction de l’image Docker

Une fois les fichiers en place, exécutez la commande suivante pour construire l'image Docker :

```
docker build -t fast_api:v0 .
```

Cette commande crée une image Docker nommée fast_api avec la version v0.

Après la construction de l’image, lancez un conteneur avec la commande :

```
docker run -p 8000:8000 fast_api:v0
```

Votre API FastAPI sera alors accessible à l'adresse : http://localhost:8000.

Testez l'API en accédant à la documentation interactive générée par FastAPI : http://localhost:8000/docs

Cette partie explique comment déployer votre projet sur Google Cloud Platform (GCP) en utilisant Google Container Registry et Cloud Run.

L'objectif est d'héberger notre image Docker dans Google Cloud Container Registry, puis de la déployer avec Cloud Run.

1. Créez un compte sur Google Cloud Platform et profitez de l'essai gratuit de 300$: Créer un compte GCP
2. Connectez-vous à la console Google Cloud.
3. Créez un nouveau projet.

Installez l'outil de ligne de commande gcloud, qui permet d'interagir avec Google Cloud depuis votre machine locale :

- Suivez les instructions officielles : Installer gcloud

Depuis le terminal, utilisez les commandes suivantes :

```
gcloud init
```

Cette commande permet de :

- Se connecter à son compte Google Cloud
- Définir le projet sur lequel on travaille
- Créer une nouvelle configuration (ex. fastapi)
- Sélectionner le projet précédemment créé

gcloud auth configure-docker

Cette commande permet d'autoriser Docker à se connecter à Google Container Registry.

Avant de pousser notre image Docker, nous devons la renommer avec le format attendu par Google Cloud :

```
docker tag fast_api:v0 gcr.io/<nom_du_projet>/fast_api:v0
```

Remplacez <nom_du_projet> par le nom de votre projet GCP.

```
docker push gcr.io/<nom_du_projet>/fast_api:v0
```

1. Allez sur Google Cloud Console
2. Activez l'API **Google Container Registry**
3. Accédez au **Container Registry** depuis la console

1. Ouvrez **Cloud Run** dans Google Cloud Console.
2. Créez un **nouveau service**.
3. Sélectionnez l'image Docker que vous venez de pousser.
4. Configurez les options :
    - Région : Sélectionnez la région la plus proche
    - Autorisations : Définissez les autorisations requises
    - Spécifiez le port **8000**
5. Cliquez sur **Créer**.

Une fois l'application déployée, vous obtiendrez une **URL publique**. Vous pourrez accéder à votre API FastAPI via cette URL.