Université Paris Cité
UFR Mathématiques Informatique

MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

Documentation Interne

Projet informatique L3E1

Université Paris Cité

2024-2025

Indexation d'une base d'images pour le classement et la recherche d'images par le contenu : utilisation de deep features issus de réseaux de neurones

Cette documentation interne correspond au projet d'informatique L3E1 :

"Indexation d'une base d'images pour le classement et la recherche d'images par le contenu : utilisation de deep features issus de réseaux de neurones"

Auteurs :

Belguedj Nassilya
Aidoudi Aaron

Moustache Mathieu

Tuteur de projet : M. Camille Kurtz

Version 1.0 du : 14/04/2025

Contents

1 Introduction

Ce document constitue la documentation interne de l'application, conçue pour faciliter sa maintenance, son évolution, ainsi que sa compréhension par des développeurs tiers. Il vise à décrire précisément la structure logicielle du projet, les modules utilisés, leurs responsabilités, leurs dépendances, ainsi que les interfaces entre eux.

L'objectif principal est de fournir une vue claire et détaillée de l'architecture du projet, des fonctions clés exposées par chaque module, et des types de données échangés, afin de permettre une prise en main rapide et efficace par une équipe de maintenance.

Cette documentation interne est complémentaire aux outils de génération automatique (comme la documentation intégrée dans les modules Python via des docstrings). Elle va plus loin en présentant également :

- La description des modules constituant le projet (e.g., extraction de caractéristiques, recherche d'images, interface Streamlit, etc.)
- Les relations de dépendance entre modules
- Les fonctions visibles depuis l'extérieur et les structures de données importantes

L'application repose principalement sur deux approches de recherche :

- **CBIR (Content-Based Image Retrieval)** : à partir d'une image, extraire ses caractéristiques visuelles via un modèle CNN (MobileNetV3), et rechercher les images les plus similaires à l'aide d'un index FAISS.
- **TBIR (Text-Based Image Retrieval)** : à partir d'une requête textuelle, encoder le texte via CLIP, et rechercher les images visuellement associées selon des embeddings pré-calculés.

L'ensemble du projet est développé en `Python 3`, utilise diverses bibliothèques indiquée dans ce document, et repose sur un déploiement simple via `Ngrok` pour des raisons de praticité et de gestion de fichiers volumineux.

Cette documentation est conçue pour évoluer avec le projet et être enrichie à mesure que de nouvelles fonctionnalités sont ajoutées ou que la structure est modifiée. Elle constitue un support essentiel à la transmission du projet dans le cadre d'une reprise de maintenance ou de poursuite de développement.

2 Préambule

2.1 Objectifs et méthodes

Ce projet repose sur l'exploitation de l'apprentissage profond pour l'indexation d'images et la recherche par similarité. Les objectifs sont multiples :

- Assurer une indexation efficace des images en extrayant des caractéristiques visuelles à l'aide de modèles pré-entraînés tels que MobileNet.
- Implémenter une recherche rapide d'images similaires en utilisant des algorithmes d'approximation des plus proches voisins via **FAISS**.
- Développer une interface utilisateur intuitive permettant aux utilisateurs de téléverser une image et d'obtenir des résultats pertinents en quelques secondes.
- Tester et valider chaque module du projet à l'aide de stratégies de validation rigoureuses, incluant des tests unitaires et d'intégration.

La méthodologie adoptée repose sur le modèle Agile, permettant des itérations fréquentes avec des tests et des ajustements progressifs.

2.2 Documents de référence

Les documents et ressources référencés pour la validation du projet incluent :

- **Le cahier des charges** du projet L3E1, qui définit les exigences initiales et les fonctionnalités attendues.
- **Le cahier de recette** du projet L3E1, détaillant les principes fondamentaux du projet.
- **Le plan de développement** définissant les modules du système, leur architecture et leur interaction, et plus largement le plan que suivront les développeurs lors de l'implémentation.
- **Le plan de tests** garantissant que le logiciel respecte le cahier des charges.
- **Documentation des outils utilisés** :
  - **PyTorch** : utilisé comme framework principal pour le deep learning. Il a permis de charger, manipuler et exploiter efficacement les modèles de réseaux de neurones convolutifs, notamment pour l'extraction des descripteurs visuels à partir des images.
  - **FAISS (Facebook AI Similarity Search)** : bibliothèque développée par Facebook AI, utilisée pour l'indexation rapide et la recherche de similarité entre vecteurs. Elle permet de comparer efficacement les vecteurs extraits pour retrouver les images les plus similaires à une image requête dans la base.
  - **MobileNet** : modèle de CNN (Convolutional Neural Network) léger et optimisé, choisi pour extraire les caractéristiques visuelles des images. Son efficacité en termes de performance et de légèreté le rend adapté pour des systèmes temps réel ou déployables sur des infrastructures légères.

  - **Streamlit** : framework web Python utilisé pour développer rapidement une interface utilisateur interactive. Il permet de présenter les résultats de recherche d'images, d'uploader de nouvelles images requêtes et d'ajuster les paramètres de l'algorithme.
  - **CLIP (Contrastive Language–Image Pretraining)** : modèle multimodal développé par OpenAI, utilisé ici pour la fonctionnalité TBIR (Text-Based Image Retrieval). Il permet de faire correspondre des requêtes textuelles avec les images les plus pertinentes, en encodant texte et images dans un même espace sémantique. Cette fonctionnalité a été ajoutée comme extension optionnelle en fin de développement.

- **Références académiques** sur l'apprentissage profond appliqué à la vision par ordinateur.

L'ensemble de ces documents garantit une approche rigoureuse dans le développement du projet et la validation des résultats obtenus.

3 Guide de lecture

Ce document est destiné aux développeurs et mainteneurs du projet, en leur fournissant une vue d'ensemble claire de l'architecture logicielle et des modules principaux composant l'application. Il a pour objectif de faciliter la compréhension du fonctionnement interne du système, d'assurer une prise en main rapide du code, et de permettre des interventions efficaces en cas de maintenance ou d'évolution future.

Chaque module est décrit individuellement, en détaillant :

- son objectif principal,
- ses dépendances avec d'autres modules,
- les fonctions qu'il expose,
- les types de données qu'il utilise ou retourne.

Ce document doit être utilisé comme point d'entrée pour comprendre la structure interne du projet, en complément des commentaires dans le code source. Il permet également de vérifier que la structure logicielle est alignée avec les besoins exprimés dans le cahier des charges et avec les choix techniques réalisés tout au long du développement.

4 Concepts de base

Le projet L3E1 repose sur l'intégration de concepts avancés en vision par ordinateur, intelligence artificielle et recherche d'information. Cette section introduit les fondements théoriques nécessaires à la compréhension de l'implémentation logicielle, en particulier la représentation vectorielle des images et la recherche de similarité dans des espaces de grande dimension.

4.1 Recherche par contenu visuel (CBIR)

Le Content-Based Image Retrieval (CBIR) permet de retrouver des images similaires en se basant uniquement sur leur contenu visuel (formes, couleurs, textures), sans métadonnées. L'approche repose sur l'extraction automatique de descripteurs à partir des images, lesquels sont ensuite utilisés pour effectuer une comparaison dans un espace vectoriel.

4.2 Réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutifs sont au cœur du système d'extraction de caractéristiques visuelles. Des modèles pré-entraînés comme **MobileNetV3** sont utilisés pour produire, à partir d'une image, un vecteur dense représentant sa signature visuelle. Ce vecteur résume les éléments visuels discriminants de l'image et permet la recherche par similarité.

Les principales étapes incluent :

1. Prétraitement (redimensionnement, normalisation) de l'image.
2. Passage dans un CNN pour obtenir un vecteur de caractéristiques.
3. Stockage et indexation de ces vecteurs.

4.3 Indexation avec FAISS

Afin de rendre la recherche rapide et scalable, le système s'appuie sur **FAISS** (Facebook AI Similarity Search), une bibliothèque conçue pour la recherche de plus proches voisins dans des ensembles massifs de vecteurs.

FAISS permet :

- D'indexer efficacement des millions de vecteurs de haute dimension.
- D'optimiser la mémoire via des méthodes de quantification.
- De répondre à des requêtes de recherche avec un temps de latence très faible.

4.4 Recherche par texte (TBIR)

Le système prend également en charge la recherche d'image à partir d'une requête textuelle. Cette fonctionnalité repose sur le modèle **CLIP** (Contrastive Language–Image Pretraining), développé par OpenAI. CLIP encode les textes et les images dans un espace vectoriel partagé, permettant une correspondance sémantique entre le langage et les visuels.

5 Architecture du système

L'architecture logicielle du projet a été conçue de manière modulaire afin de faciliter le développement, les tests, l'extension et la maintenance du système. L'application repose sur six modules principaux, interconnectés selon un pipeline de traitement clair :

- **Prétraitement des images**
  Ce module s'occupe de préparer les images utilisateur pour leur traitement par les réseaux neuronaux : redimensionnement, conversion de format, normalisation, etc. Une uniformité de prétraitement est essentielle pour garantir la cohérence des vecteurs extraits.

- **Extraction des caractéristiques (MobileNetV3)**
  Utilisation de MobileNetV3 via PyTorch pour générer un vecteur de caractéristiques par image. Ce vecteur est un résumé visuel compact et discriminant de l'image.

- **Indexation et recherche (FAISS)**
  Les vecteurs extraits sont indexés dans FAISS pour permettre une recherche ultra-rapide de plus proches voisins. Le module interagit avec le module d'extraction et celui de requêtes utilisateur.

- **Base de données locale**
  Contient les images du dataset Tiny ImageNet ainsi que leurs descripteurs visuels pré-calculés. Cela permet d'accélérer considérablement le temps de traitement au moment des recherches.

- **Interface utilisateur (Streamlit)**
  Interface web légère permettant de téléverser une image, lancer une recherche, visualiser les résultats. Ce module communique directement avec le backend pour récupérer les résultats de similarité.

- **Recherche textuelle (CLIP)**
  Intégration de CLIP pour le Text-Based Image Retrieval. Ce module convertit une requête textuelle en vecteur, qu'il compare ensuite aux vecteurs d'image dans l'espace CLIP. Il partage certaines structures de données avec les modules de recherche CBIR.

L'architecture globale peut être représentée sous forme de pipeline :

[Image utilisateur ou texte] → *[Prétraitement]* → *[Extraction / Encodage]* → *[Recherche FAISS]* → *[Résultats]* → *[Affichage Streamlit]*

Cette structuration modulaire assure une bonne séparabilité des responsabilités, facilitant la maintenance et la réutilisation de chaque composant dans d'autres projets ou versions futures du système.

L'architecture globale peut aussi être représentée sous forme de schéma:

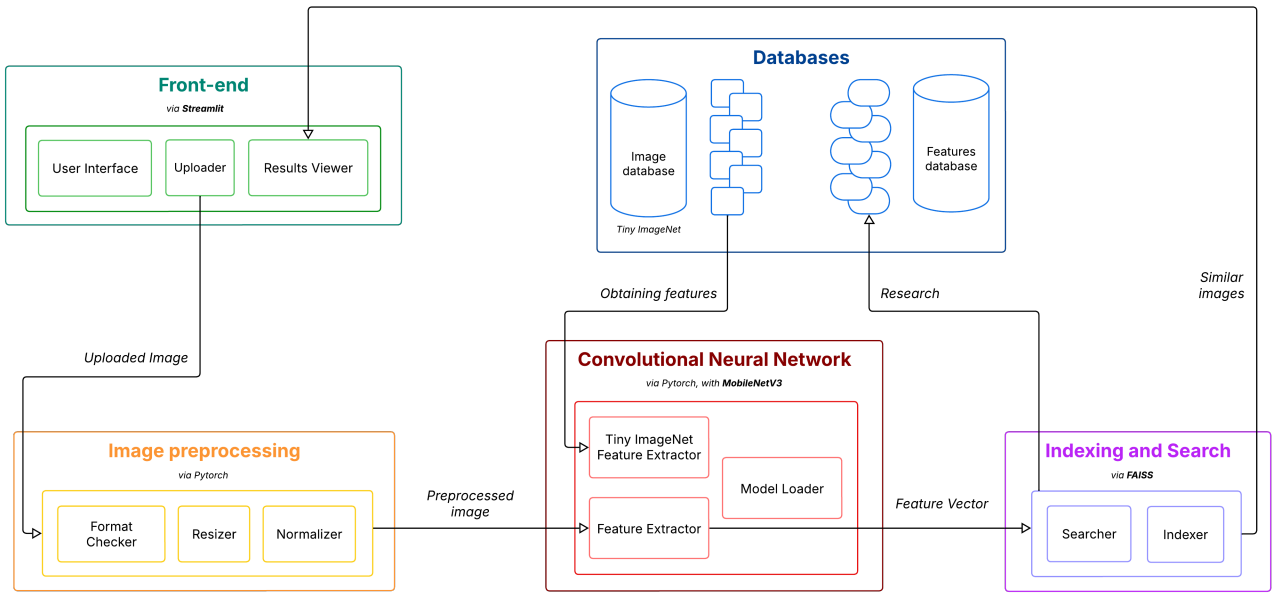

Figure 1: Schéma de l'architecture globale du système

6 Architecture Générale du Projet

Le projet est organisé en plusieurs répertoires logiques afin de séparer clairement les différentes responsabilités du système : code source, ressources de données, interface utilisateur, et tests. Cette organisation facilite la maintenance, la lisibilité et l'évolutivité du code. Ci-dessous, une vue détaillée de l'architecture des fichiers et répertoires utilisés.

6.1 Arborescence du projet

6.2 Commentaires sur la structure

- Le répertoire `src/` regroupe tous les modules fonctionnels. Chaque fichier source y correspond à une étape clé du pipeline de traitement (prétraitement, extraction, recherche, interface).
- Le dossier `ressources/` contient les fichiers nécessaires à l'indexation et à la recherche. Les fichiers `*.npy` sont les résultats des traitements d'extraction, et permettent d'éviter leur recalcul à chaque lancement. Tous les fichiers contenus dans ce dossier (à part ceux `*.py`) ne sont pas inclus directement dans le rendu final du fait de la taille de ceux-ci, ils sont à télécharger par la suite. Plus de détails dans le `README.md`. Aussi, une amélioration a été ajoutée en fin de développement sur le dataset : l'extraction des caractéristiques a été faite sur un nouveau, Open Images, permettant une plus grande diversité de recherches et des images en meilleure résolution.
- Le sous-répertoire `frontend/`, contenu dans `src/`, contient l'interface utilisateur de l'application, développée avec **Streamlit**. Streamlit est un framework Python léger

```
PixMatcher/
│
├── requirements.txt            # List of dependencies
├── README.txt                  # Project documentation
│
├── src/                        # Main application source code folder
│   ├── __init__.py             # Marks the folder as a Python package
│   │
│   ├── image_preprocessing.py  # Image preprocessing module
│   ├── feature_extractor.py    # Feature extraction module, with MobileNetV3
│   ├── similarity_search.py    # Similar image search module, with FAISS and in the Tiny ImageNet or Open Images datasets
│   │
│   ├── clip_similarity_search.py  # Similar image search module via text, in the Tiny ImageNet or Open Images datasets
│   │
│   ├── frontend/               # Web interface via Streamlit
│   │   ├── main_frontend.py    # Main page
│   │   ├── research.py         # CBIR research page
│   │   ├── clip_research.py    # TBIR research page
│   │   ├── visualization.py    # Visualization page
│   │   ├── about.py            # About page
│
├── ressources/                 # Folder of embeddings, categories, images and links of the two datasets
│   ├── tiny-imagenet/          # Tiny ImageNet folder
│   │   ├── tiny-imagenet-200   # Tiny ImageNet dataset, containing 100k images (to download)
│   │   │
│   │   ├── tinyimagenet_mobilnetv3_feature_extractor.py  # Extractor of vectors from the dataset with MobileNetV3 (for CBIR)
│   │   ├── Tiny_ImageNet_MobilNetV3_Categories.npy       # Image related categories (for CBIR / to download)
│   │   ├── Tiny_ImageNet_MobilNetV3_Embeddings.npy       # Image-related feature vectors (for CBIR / to download)
│   │   │
│   │   ├── tinyimagenet_clip_feature_extractor.py  # Extractor of vectors from the dataset with CLIP (for TBIR)
│   │   ├── Tiny_ImageNet_CLIP_Categories.npy       # Image related categories (for TBIR / to download)
│   │   ├── Tiny_ImageNet_CLIP_Embeddings.npy       # Image-related feature vectors (for TBIR / to download)
│   │
│   ├── open-images/                # Open Images folder
│   │   ├── images_urls.json        # Table of links pointing to each of the images in Open Images
│   │   │
│   │   ├── clip_embeddings.npy     # Image-related feature vectors (for TBIR / to download)
│   │   ├── mobilenet_embeddings.npy  # Image-related feature vectors (for CBIR / to download)
│
├── test/                       # Unit tests for modules
│   ├── app_test.py             # Test for all modules
│   ├── image_preprocessing_test.py
│   ├── feature_extractor_test.py
│   ├── similarity_search_test.py
│
```

Figure 2: Arborescence de l'application

qui permet de créer des applications web interactives en quelques lignes de code. Il est particulièrement adapté aux projets de data science ou de machine learning car il facilite l'intégration directe de composants Python, comme des graphiques, des formulaires ou des visualisations.
L'interface développée permet notamment :

- Le téléversement d'une image ou la saisie d'une requête textuelle.
- Le déclenchement d'une recherche CBIR (Content-Based Image Retrieval) ou TBIR (Text-Based Image Retrieval).
- L'affichage dynamique des images similaires retournées par le système.

Grâce à Streamlit, l'utilisateur peut interagir avec l'application de façon fluide sans avoir à recharger manuellement la page ou à manipuler des scripts complexes.

- `test/` : Ce répertoire regroupe les tests unitaires et les tests d'intégration, réalisés à l'aide de la bibliothèque **unittest** intégrée à Python. Ces tests sont essentiels pour assurer la robustesse et la fiabilité de chaque composant du système. Ils permettent de :
  - Valider le bon comportement de chaque module indépendamment (tests unitaires).
  - Vérifier la cohérence du système dans son ensemble (tests d'intégration).
  - Détecter rapidement les régressions ou erreurs lors des modifications de code.

Chaque fichier de test cible un module spécifique (par exemple, `feature_extractor_test.py` teste les fonctions du module `feature_extractor.py`). Le fichier `app_test.py`, quant à lui, vérifie le bon enchaînement de tous les modules dans un scénario complet.
Pour une description exhaustive des cas de test et de leur rôle dans le processus de validation, se référer au **plan de test** associé.

- `requirements.txt` : Ce fichier liste de manière exhaustive toutes les bibliothèques Python nécessaires au bon fonctionnement du projet. Il permet de reproduire fidèlement l'environnement d'exécution à l'aide de la commande `pip install -r requirements.txt`.
  Son rôle est crucial, notamment dans le contexte de ce projet, qui repose sur un grand nombre de dépendances spécifiques (PyTorch, Streamlit, FAISS, Transformers, NumPy, etc.), parfois avec des versions précises requises pour garantir la compatibilité.
  Il permet aussi :
  - De faciliter le déploiement sur un autre environnement ou serveur.
  - D'éviter les conflits de dépendances ou les erreurs d'exécution dues à des versions incompatibles.
  - De garantir que tout développeur ou mainteneur du projet dispose du même environnement de développement.

7 Module `image_preprocessing.py`

7.1 Objectif du module

Ce module assure le **prétraitement des images** en amont de leur passage dans le réseau de neurones. Il garantit que les images sont au bon format, de taille adaptée et prêtes à être converties en tenseurs PyTorch pour l'extraction de caractéristiques. Il joue donc un rôle fondamental dans la standardisation des données d'entrée du système CBIR.

7.2 Modules utilisés

- `os` : Pour la manipulation des chemins de fichiers.
- `logging` : Pour journaliser les opérations et les éventuels avertissements.
- `PIL.Image` : Pour ouvrir, convertir et redimensionner les images.
- `torchvision.transforms` : Pour convertir les images PIL en tenseurs normalisés.

7.3 Utilisé par

- `feature_extractor.py` : Utilise la fonction `preprocess_image()` pour préparer les images avant leur passage dans MobileNetV3.
- `frontend/app.py` : Pour préparer l'image uploadée par l'utilisateur.

7.4 Définitions de types / Attributs

- `ALLOWED_EXTENSIONS` (`list[str]`) : Liste des extensions de fichiers images acceptées.
- `InvalidImageFormatError` (`Exception`) : Exception personnalisée levée lorsqu'un format non pris en charge est détecté.

7.5 Procédures externes (fonctions visibles)

- `is_allowed_extension(file_path: str)` $\rightarrow$ `bool` : Vérifie si l'extension du fichier est valide.
- `preprocess_image(image_path: str, target_size: tuple, max_dim: int, to_tensor: bool)` $\rightarrow$ `Union[PIL.Image, torch.Tensor]`
  Fonction principale du module. Elle ouvre une image, la convertit au format RGB si nécessaire, redimensionne l'image à la taille cible et retourne soit une image PIL, soit un tenseur PyTorch prêt à être passé à un CNN.

8 Module `feature_extractor.py`

8.1 Objectif du module

Ce module encapsule le processus d'extraction de caractéristiques (features) d'une image à l'aide du modèle **MobileNetV3 pré-entraîné** sur ImageNet. Il fournit une classe `FeatureExtractor` capable de transformer une image en un vecteur de caractéristiques (embedding) utilisable pour la recherche d'images similaires dans le système CBIR.

8.2 Modules utilisés

- `torch, torchvision` : Chargement du modèle, manipulation des tenseurs, inférence.
- `PIL.Image` : Pour ouvrir les images (en cas de fallback).
- `image_preprocessing` : Pour le prétraitement avancé des images (si disponible).
- `numpy` : Pour convertir les tenseurs en vecteurs manipulables facilement.
- `logging` : Pour signaler les avertissements (notamment si le module de prétraitement est introuvable).

8.3 Utilisé par

- `similarity_search.py` : Pour extraire les caractéristiques de l'image requête.
- `frontend/app.py` : Pour extraire les features de l'image uploadée par l'utilisateur.

8.4 Définitions de types / Attributs

- `FeatureExtractor.device` : Device sur lequel les calculs sont effectués (`cpu` ou `cuda`).
- `FeatureExtractor.feature_extractor` : Pipeline tronqué de MobileNetV3 (sans couche de classification).
- `FeatureExtractor.basic_transform` : Pipeline de transformation alternatif si `image_preprocessing` est indisponible.

8.5 Procédures externes (méthodes publiques)

- `FeatureExtractor(device: str = None, target_size: tuple[int, int] = (224, 224))`
  Constructeur. Initialise MobileNetV3, sélectionne le bon `device`, définit la taille cible et le pipeline d'extraction.
- `extract_features(image_input, from_preprocessed: bool = False)` $\rightarrow$ `np.ndarray`
  Méthode principale. Extrait un vecteur de caractéristiques 1D à partir d'une image (chemin ou objet PIL). Gère les cas d'image déjà prétraitée ou non, utilise le module `image_preprocessing` si disponible.

9 Module `similarity_search.py`

9.1 Objectif du module

Ce module gère la **recherche de similarité d'images** dans plusieurs datasets, notamment **Tiny ImageNet** et, depuis la dernière amélioration, **Open Images**. Il permet de retrouver les images ou catégories les plus proches d'une image requête, en utilisant un moteur de recherche vectorielle basé sur `FAISS` (Facebook AI Similarity Search). Le système s'appuie sur des vecteurs d'embeddings extraits par MobileNetV3. L'ajout du support d'Open Images a été réalisé en fin de développement comme une amélioration pour élargir les possibilités de recherche et de comparaison d'images, ainsi que l'amélioration de la qualité (résolution) des images.

9.2 Modules utilisés

- `numpy` : Manipulation des tableaux d'embeddings.
- `faiss` : Construction de l'index vectoriel pour recherche rapide.
- `scipy.spatial.distance` : Calcul des distances cosinus pour les catégories.
- `os, pathlib` : Gestion des chemins de fichiers et ressources.
- `json` : Chargement des mappings d'images pour Open Images.

9.3 Utilisé par

- `frontend/app.py` : Pour afficher les résultats de la recherche d'images ou de catégories similaires, au choix de l'utilisateur sur Tiny ImageNet ou Open Images.

9.4 Définitions de types / Attributs

- `TINY_IMAGENET_PATH` : Chemin vers le dataset Tiny ImageNet (répertoire racine).
- `TI_EMBEDDINGS_PATH` : Fichier contenant les embeddings Tiny ImageNet pré-calculés (fichier numpy).
- `TI_CATEGORIES_PATH` : Fichier contenant la catégorie associée à chaque embedding Tiny ImageNet.
- `OI_EMBEDDINGS_PATH` : Fichier contenant les embeddings Open Images (fichier numpy).
- `image_urls` : Mapping index → nom d'image pour Open Images (les images n'étant pas stocké localement mais sur un serveur accessible via internet.
- `oi_index`, `ti_index` : Index FAISS de dimension fixe pour chaque dataset, construits au chargement du module.

9.5 Procédures externes (fonctions publiques)

- `ti_find_top5_categories(image`$_{f}$*eatures* : *np.ndarray*)→ `list[str, float]` Retourne les 5 catégories les plus proches d'un vecteur d'image dans Tiny ImageNet, en utilisant la distance cosinus. La correspondance des identifiants (WordNet ID) avec leurs noms est lue depuis le fichier `words.txt`.

- `ti_find_top_similar_images(image_features: np.ndarray, k: int)` → `list[tuple]` Lance une recherche FAISS pour retrouver les `k` images les plus similaires dans Tiny ImageNet.

- `ti_get_image_path(index_image: int, base_path = TINY_IMAGENET_PATH)` → `str`
  Reconstitue le chemin d'accès à une image du dataset Tiny ImageNet, à partir de son index dans le fichier d'embeddings.

- `oi_find_top_similar_images(image_features: np.ndarray, k: int)` → `list[tuple]` Recherche les `k` images les plus similaires dans Open Images à partir d'un vecteur d'embedding, via FAISS.

- `oi_get_image_path(index_image: int)` → `str`
  Retourne l'URL complète d'une image Open Images à partir de son index, en s'appuyant sur le mapping JSON fourni.

Remarque : L'ajout du support d'Open Images permet désormais d'effectuer des recherches de similarité sur deux jeux de données distincts, offrant ainsi une plus grande flexibilité et une couverture d'images plus large pour l'utilisateur. Cette extension a été intégrée en fin de développement comme une amélioration significative du module.

10 Module `tinyimagenet_mobilnetv3_feature_extractor.py`

10.1 Objectif du module

Ce module permet d'**extraire les vecteurs de caractéristiques** (embeddings) du jeu de données Tiny ImageNet à l'aide du modèle MobileNetV3. Les étapes suivantes sont réalisées :

- Téléchargement et prétraitement des images du dataset Tiny ImageNet.
- Chargement du modèle MobileNetV3 avec suppression de la dernière couche pour obtenir les embeddings intermédiaires.
- Extraction des embeddings par lots pour optimiser les performances.
- Sauvegarde des embeddings et des catégories dans des fichiers numpy pour un usage ultérieur.

Le programme est conçu pour être exécuté sur Google Colab et prend environ 1 heure pour traiter l'ensemble du dataset.

10.2 Modules utilisés

- `torch` : Utilisé pour charger et manipuler le modèle MobileNetV3 ainsi que pour extraire les caractéristiques des images.
- `numpy` : Manipulation des vecteurs d'embeddings et sauvegarde sous format `.npy`.
- `os, shutil, zipfile` : Pour la gestion des fichiers et l'extraction du dataset Tiny ImageNet.
- `PIL` : Pour ouvrir et transformer les images.
- `requests` : Pour télécharger le dataset depuis une URL.
- `torchvision.models` : Pour charger le modèle MobileNetV3 pré-entraîné.
- `torchvision.transforms` : Pour prétraiter les images avant leur passage dans le modèle.
- `tqdm` : Pour afficher une barre de progression lors de l'extraction des embeddings.

10.3 Définitions de types / Attributs

- `device` : Définit le périphérique (GPU ou CPU) utilisé pour l'exécution du modèle.
- `TINY_IMAGENET_URL` : URL du fichier zip contenant le dataset Tiny ImageNet.
- `DATASET_PATH` : Chemin vers le dossier où le dataset sera extrait.
- `model` : Le modèle MobileNetV3 pré-entraîné utilisé pour l'extraction des embeddings.

- `feature_extractor` : Version du modèle MobileNetV3 sans la dernière couche, utilisée pour extraire les embeddings.
- `transform` : Séries de transformations appliquées aux images avant de les envoyer dans le modèle (redimensionnement, normalisation).
- `embeddings` : Liste contenant les embeddings extraits pour chaque image du dataset.
- `categories` : Liste contenant les catégories associées à chaque image (dossier d'image).
- `batch_size` : Taille du lot d'images pour l'extraction par lots.

10.4 Procédures externes (fonctions publiques)

- `extract_features_batch(batch: torch.Tensor)` → `np.ndarray`
  Extrait les embeddings d'un lot d'images en utilisant le modèle MobileNetV3 sans la dernière couche. Les caractéristiques sont ensuite transformées en vecteurs unidimensionnels et renvoyées sous forme de tableau numpy.

10.5 Exécution sur Google Colab

Ce programme est conçu pour être exécuté sur Google Colab, avec les étapes suivantes :

- Téléchargement automatique du dataset Tiny ImageNet.
- Extraction des embeddings des images par lots.
- Sauvegarde des embeddings sous forme de fichiers `.npy`.
- Téléchargement optionnel des fichiers résultants sur l'ordinateur local via Google Colab.

11 Module `clip_similarity_search.py`

11.1 Objectif du module

Ce module permet de **rechercher des images similaires à une requête textuelle** en utilisant le modèle CLIP de OpenAI. CLIP (Contrastive Language-Image Pretraining) est capable de relier du texte et des images dans un espace vectoriel commun. Le module charge les embeddings d'images Tiny ImageNet et utilise la similarité cosinus pour trouver les images les plus pertinentes par rapport à une description textuelle donnée.

11.2 Modules utilisés

- `torch` : Utilisation de PyTorch pour l'encodage du texte et la gestion du modèle CLIP.
- `clip` : Le modèle CLIP d'OpenAI pour l'encodage du texte et des images.
- `numpy` : Manipulation des embeddings d'images et calcul des distances.
- `scipy.spatial.distance` : Calcul de la distance cosinus entre les vecteurs.
- `os, pathlib` : Gestion des chemins de fichiers et accès aux ressources du dataset.

11.3 Utilisé par

- `frontend/app.py` : Pour exécuter des requêtes textuelles et afficher les résultats des recherches d'images.

11.4 Définitions de types / Attributs

- `device` : "cuda" si un GPU est disponible, sinon "cpu". Utilisé pour déterminer où charger le modèle CLIP.
- `model, preprocess` : Modèle CLIP chargé avec les poids pré-entraînés.
- `EMBEDDINGS_PATH` : Fichier contenant les embeddings d'images Tiny ImageNet calculés à l'aide de CLIP.
- `CATEGORIES_PATH` : Fichier contenant les catégories associées aux embeddings.
- `BASE_PATH` : Chemin d'accès au dataset Tiny ImageNet.

11.5 Procédures externes (fonctions publiques)

- `text_to_vector(text: str)` → `np.ndarray`
  Encode une requête textuelle en un vecteur d'embedding. Le texte est tokenisé et passé dans le modèle CLIP. Le vecteur est ensuite normalisé.
- `find_similar_images(text: str, top_k: int = 10)` → `np.ndarray`
  Trouve les `top_k` images les plus similaires à une requête textuelle en calculant la distance cosinus entre le vecteur d'embedding du texte et ceux des images.

- `get_image_path(index: int) → str`
  Récupère le chemin complet d'une image à partir de son index dans le fichier d'embeddings. Cette fonction permet de retrouver l'image associée à l'index retourné par la fonction `find_similar_images`.

12 Module `tinyimagenet_clip_feature_extractor.py`

12.1 Objectif du module

Ce module utilise le modèle CLIP (ViT-B/32) pour extraire les embeddings d'images du dataset Tiny ImageNet. Les étapes suivantes sont réalisées :

- Téléchargement automatique du dataset Tiny ImageNet si nécessaire.
- Chargement du modèle CLIP (ViT-B/32).
- Extraction des embeddings par lots afin d'optimiser les performances.
- Sauvegarde des embeddings et des catégories associées dans des fichiers numpy (.npy).

Le programme est conçu pour être exécuté sur Google Colab et prend environ 5 heures pour traiter l'ensemble du dataset.

12.2 Modules utilisés

- `torch` : Utilisé pour charger et manipuler le modèle CLIP ainsi que pour l'extraction des embeddings des images.
- `clip` : Pour charger le modèle CLIP et effectuer l'encodage des images en embeddings.
- `numpy` : Manipulation des vecteurs d'embeddings et sauvegarde sous format `.npy`.
- `os, shutil, zipfile` : Pour la gestion des fichiers et l'extraction du dataset Tiny ImageNet.
- `requests` : Pour télécharger le dataset depuis une URL.
- `PIL` : Pour ouvrir et transformer les images.
- `tqdm` : Pour afficher une barre de progression lors de l'extraction des embeddings.

12.3 Utilisé par

- `test/test_tinyimagenet_clip_feature_extractor.py` : Pour tester l'extraction des embeddings avec CLIP.
- `backend/clip_feature_extraction_service.py` : Pour intégrer l'extraction des embeddings dans un pipeline de traitement d'images.

12.4 Définitions de types / Attributs

- `device` : Définit le périphérique (GPU ou CPU) utilisé pour l'exécution du modèle.
- `TINY_IMAGENET_URL` : URL du fichier zip contenant le dataset Tiny ImageNet.
- `DATASET_PATH` : Chemin vers le dossier où le dataset sera extrait.
- `model` : Le modèle CLIP ViT-B/32 chargé pour l'extraction des embeddings.
- `preprocess` : La fonction de prétraitement appliquée aux images avant de les envoyer dans le modèle CLIP.
- `embeddings` : Liste contenant les embeddings extraits pour chaque image du dataset.
- `categories` : Liste contenant les catégories associées à chaque image (dossier d'image).
- `batch_size` : Taille du lot d'images pour l'extraction par lots.
- `batch_images` : Liste temporaire pour accumuler les images d'un lot avant traitement.
- `batch_categories` : Liste temporaire pour accumuler les catégories correspondantes aux images d'un lot.

12.5 Procédures externes (fonctions publiques)

- `extract_features_batch(batch: torch.Tensor)` $\rightarrow$ `np.ndarray`
  Extrait les embeddings d'un lot d'images en utilisant le modèle CLIP. Les caractéristiques sont ensuite normalisées et renvoyées sous forme de tableau numpy.

12.6 Exécution sur Google Colab

Ce programme est conçu pour être exécuté sur Google Colab, avec les étapes suivantes :

- Téléchargement automatique du dataset Tiny ImageNet si nécessaire.
- Extraction des embeddings des images par lots.
- Sauvegarde des embeddings sous format `.npy`.
- Affichage du temps d'exécution total et du nombre d'images traitées.

Les fichiers résultants (`Tiny_ImageNet_CLIP_Embeddings.npy` et `Tiny_ImageNet_CLIP_Categories.npy`) sont sauvegardés sur le disque local de Google Colab.

13 Module Web (`/frontend`)

Le module **Streamlit** est utilisé ici pour créer une interface utilisateur interactive et dynamique, principalement pour la gestion des différentes pages de l'application (comme la recherche par image, la recherche par texte, la visualisation, etc.). Il permet de développer rapidement des applications web basées sur Python avec des composants UI simples à intégrer.

13.1 Fonctionnement de Streamlit

- **Navigation avec `option_menu`** : L'application utilise la bibliothèque `streamlit_option_menu` pour créer un menu de navigation sur le côté de l'interface. Ce menu permet de naviguer entre différentes sections comme la recherche par image, la recherche par texte, la visualisation des résultats, et l'affichage des informations à propos de l'application.

- **Chargement de la page sélectionnée** : En fonction de la page sélectionnée par l'utilisateur dans le menu, Streamlit importe dynamiquement et exécute le fichier approprié, comme `research.py` pour la recherche par image ou `clip_research.py` pour la recherche par texte. Cela permet de séparer le code des différentes sections tout en restant dans une même interface.

- **Fonctionnement de `research.py`** : Dans le fichier `research.py`, le code gère principalement l'upload d'une image par l'utilisateur, l'extraction des caractéristiques de l'image téléchargée à l'aide du modèle AI, et la recherche des images les plus similaires dans la base de données. Le processus suit ces étapes :

  1. L'utilisateur télécharge une image via un `file_uploader`.
  2. L'image est ensuite traitée et les caractéristiques sont extraites par un modèle de `FeatureExtractor`.
  3. Une recherche est effectuée pour identifier les images les plus similaires basées sur des critères de similarité visuelle, comme la texture, la couleur, et la forme.
  4. Les résultats sont ensuite affichés sous forme d'images similaires à l'utilisateur.

- **UI et Esthétique** : Streamlit permet également de personnaliser l'interface via du code HTML et CSS intégré, ce qui donne une apparence plus professionnelle à l'application. Des styles sont ajoutés pour la mise en page des titres, des boutons, des étapes et pour l'upload des images, ainsi que pour les résultats de la recherche. Toute l'esthétique est basée sur la maquette travaillée lors du début de la conception du projet.

13.2 Principaux Composants Streamlit Utilisés

- `st.sidebar` : Gère l'affichage du menu latéral de navigation.

- `st.file_uploader` : Permet à l'utilisateur de télécharger une image pour la recherche.

- `st.columns` : Crée une disposition en colonnes pour afficher les résultats de manière plus fluide.

- `st.image` : Affiche une image sur l'interface utilisateur.

- `st.markdown` : Permet d'injecter du texte formaté en HTML et CSS pour personnaliser l'interface.
- `st.error` : Affiche des messages d'erreur en cas de problème lors du traitement des images.

13.3 Ngrok

Dans le cadre de ce projet, nous avons utilisé **Ngrok** pour exposer l'application web localement sur Internet de manière sécurisée. Ngrok permet de créer un tunnel sécurisé entre une application locale et le web, ce qui est particulièrement utile pour tester des applications en développement sans avoir besoin de déployer sur un serveur distant.

Nous avons choisi Ngrok car il offre une solution rapide et simple pour obtenir une URL publique pointant vers notre serveur local, facilitant ainsi le partage de l'application pendant le développement.

Fonctionnement :

1. Installez Ngrok sur votre machine.
2. Lancez votre application Streamlit localement via `streamlit run`.
3. Exécutez Ngrok avec la commande `ngrok http 8501` (port par défaut de Streamlit).
4. Ngrok vous fournira une URL publique que vous pouvez partager pour accéder à votre application depuis n'importe quel appareil.

Cette méthode est idéale pour des démonstrations ou des tests à distance pendant la phase de développement.

14 Environnement et Dépendances

Ce projet repose sur un ensemble de dépendances spécifiques pour fonctionner correctement. Ces dépendances sont listées dans le fichier `requirements.txt` et couvrent une large gamme de bibliothèques nécessaires pour le traitement des images, l'extraction de caractéristiques, la gestion de l'interface utilisateur, et les recherches basées sur la similarité. Il est essentiel de configurer correctement l'environnement avant d'exécuter le projet.

14.1 Dépendances

Les dépendances principales du projet sont les suivantes :

- `streamlit>=1.15,<2` : Bibliothèque pour la création de l'interface utilisateur interactive et dynamique.
- `numpy>=1.21` : Bibliothèque pour les calculs numériques et la gestion des tableaux multidimensionnels.
- `Pillow>=9` : Bibliothèque pour la manipulation des images.
- `torch>=2,<3` et `torchvision =0.21.0` : Bibliothèques pour l'utilisation de modèles de deep learning, notamment pour le traitement des images.
- `scipy>=1.9,<1.12` : Bibliothèque pour les opérations scientifiques et mathématiques supplémentaires.
- `requests>=2` : Utilisée pour les requêtes HTTP, en particulier pour le téléchargement des datasets.
- `tqdm>=4` : Bibliothèque pour afficher des barres de progression lors du traitement des données.
- `matplotlib>=3` : Utilisée pour l'affichage des graphiques et visualisation des résultats.
- `scikit-learn>=1` : Bibliothèque pour les outils d'analyse de données et les algorithmes de machine learning.
- `faiss-cpu` et `faiss-gpu` : Bibliothèques pour l'indexation et la recherche rapide de vecteurs d'images en utilisant FAISS, avec une prise en charge du CPU et du GPU.
- `open_clip_torch` : Bibliothèque pour l'utilisation du modèle CLIP de OpenAI, permettant l'extraction de caractéristiques d'images et de textes.

Les dépendances peuvent être installées en exécutant la commande suivante dans l'environnement virtuel Python après avoir téléchargé le fichier `requirements.txt` :

```
pip install -r requirements.txt
```

14.2 Ressources Externes

Il est important de noter que les ressources nécessaires à l'exécution du projet ne sont pas incluses directement dans le code source du projet. Certaines données doivent être téléchargées séparément.

- **Tiny ImageNet** : Un des ensembles de données utilisé pour le projet est `Tiny ImageNet`. Ce dataset, qui est un sous-ensemble d'ImageNet, contient 200 classes avec 500 images par classe. Vous pouvez télécharger le dataset depuis le lien suivant : `http://cs231n.stanford.edu/tiny-imagenet-200.zip`.
- **Open Images** : Un autre ensemble de données utilisé dans ce projet est `Open Images`. Il s'agit d'une base de données d'images à grande échelle, contenant des images annotées de manière riche et hiérarchique. Pour ce projet, un sous-ensemble filtré et redimensionné a été utilisé afin de correspondre aux besoins de la recherche d'images. Vous pouvez accéder à l'ensemble complet via le lien officiel : `https://storage.googleapis.com/openimages/web/index.html`.
- **Embeddings et Catégories** : Les fichiers d'embeddings et de catégories extraits via MobileNetV3 et CLIP sont disponibles sur le Google Drive du projet. Ces fichiers sont essentiels pour les tests et le fonctionnement de l'application.
- **MobileNetV3** : Ce modèle de classification d'images développé par Google est utilisé pour extraire les caractéristiques des images. Vous pouvez consulter la documentation de MobileNetV3 sur `https://pytorch.org/vision/stable/models/generated/torchvision.models.mobilenet_v3_small.html`.
- **CLIP - ViT-B/32** : Le modèle CLIP développé par OpenAI est utilisé pour associer des images et du texte. Vous pouvez en apprendre plus sur CLIP sur `https://openai.com/index/clip/`.
- **FAISS** : Facebook AI Similarity Search (FAISS) est utilisé pour l'indexation et la recherche rapide d'images similaires. Consultez la documentation de FAISS sur `https://faiss.ai/`.

14.3 Lancement de l'Application

Après avoir téléchargé toutes les ressources nécessaires et installé les dépendances, vous pouvez lancer l'application web avec la commande suivante :

```
streamlit run src/frontend/main_frontend.py
```

Pour tester l'application en ligne de commande, sans interface graphique, utilisez la commande suivante :

```
python3 test/app_test.py <mobilenet|clip> <image_path|query_text>
```

Assurez-vous d'avoir téléchargé les ressources nécessaires et d'avoir configuré correctement l'environnement avant de procéder à l'exécution.

15 Conclusion

En conclusion, ce projet d'indexation et de recherche d'images basé sur des caractéristiques profondes issues de modèles d'apprentissage automatique met en œuvre des technologies avancées pour résoudre des problématiques complexes en vision par ordinateur et en traitement du langage naturel. Grâce à l'utilisation de modèles performants comme MobileNetV3 et CLIP, ainsi qu'à l'intégration d'outils puissants tels que FAISS pour la recherche rapide, le système offre une solution robuste et scalable pour la recherche d'images par contenu visuel (CBIR) et par requête textuelle (TBIR).

L'approche modulaire adoptée garantit une architecture claire, facilitant la maintenance, l'évolutivité et la réutilisation des composants dans d'autres projets. L'intégration d'une interface utilisateur intuitive via Streamlit permet également une interaction fluide avec le système, rendant les fonctionnalités accessibles aussi bien aux utilisateurs techniques qu'aux non-spécialistes.

Les résultats obtenus démontrent la pertinence des choix technologiques et méthodologiques effectués tout au long du projet. Cependant, des perspectives d'amélioration existent, notamment en explorant des modèles plus récents ou en optimisant davantage les performances pour des datasets encore plus volumineux.

Ce projet constitue une base solide pour toute extension future visant à enrichir les fonctionnalités ou à adapter le système à de nouveaux domaines d'application. Il illustre également l'importance de combiner rigueur scientifique et innovation technologique dans le développement de solutions intelligentes adaptées aux besoins du monde réel.

16 Annexe

16.1 Dépendances Python

- **Streamlit** : https://docs.streamlit.io/
- **numpy** : https://numpy.org/doc/
- **Pillow (PIL)** : https://pillow.readthedocs.io/
- **torch (PyTorch)** : https://pytorch.org/docs/stable/index.html
- **torchvision** : https://pytorch.org/vision/stable/index.html
- **scipy** : https://docs.scipy.org/doc/scipy/
- **requests** : https://docs.python-requests.org/en/latest/
- **tqdm** : https://tqdm.github.io/
- **matplotlib** : https://matplotlib.org/stable/contents.html
- **scikit-learn** : https://scikit-learn.org/stable/documentation.html
- **faiss-cpu / faiss-gpu** : https://github.com/facebookresearch/faiss
- **open-clip-torch** : https://github.com/mlfoundations/open_clip

16.2 Ressources utilisées

- **Dataset Tiny ImageNet**
  Source : https://cs231n.stanford.edu/
  Lien direct : http://cs231n.stanford.edu/tiny-imagenet-200.zip
- **MobileNetV3 (PyTorch)**
  Source : https://pytorch.org/vision/stable/models/generated/torchvision.models.mobilenet_v3_small.html
- **CLIP (ViT-B/32)**
  Source : https://openai.com/index/clip/
- **FAISS - Facebook AI Similarity Search**
  Source : https://faiss.ai/
- **Ngrok (pour le déploiement temporaire)**
  Documentation : https://ngrok.com/docs
- **Fichiers Embeddings pré-extraits (Drive)**
  Téléchargement : https://drive.google.com/drive/folders/1fG2j6oRhhP7w1kNZm0svfod8yZusp=share_link