

Plan de Tests

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**

**Ce plan de tests correspond au projet d'informatique
numéro 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 2.0 du : 14/04/2025

Contents

1	Introduction	3
2	Préambule	3
2.1	Objectifs et méthodes	3
2.2	Documents de référence	3
3	Guide de lecture	4
4	Concepts de base	5
4.1	Recherche par contenu visuel (CBIR)	5
4.2	Réseaux de neurones convolutifs (CNN)	5
4.3	Indexation avec FAISS	5
4.4	Recherche par texte (TBIR)	5
5	Architecture du système	6
6	Organisation et Ordonnancement des Tests	8
6.1	Stratégie globale	8
6.2	Environnement de test	8
7	Description des Tests Unitaires	9
7.1	Module 1 – Prétraitement des Images	9
7.2	Module 2 – Extraction des Caractéristiques (CNN)	10
7.3	Module 3 – Indexation et Recherche via FAISS	10
7.4	Module 4 – Interface Utilisateur et Communication Backend	11
7.5	Module 5 – Recherche de similarité via CLIP	11
8	Description des Tests d’Intégration	13
8.1	Test d’Intégration n°1 – Recherche d’image par similarité	13
8.2	Test d’Intégration n°2 – Affichage et Interaction de l’Interface	13
8.3	Test d’Intégration n°3 – Gestion des Erreurs et Cas Particuliers	14
9	Description des Tests Fonctionnels	15
9.1	Test à Large Échelle – Évaluation Qualitative des Résultats	15
9.1.1	Recherche par Image (CBIR)	15
9.1.2	Recherche par Texte (TBIR avec CLIP)	16
9.2	Test Fonctionnel – Fonctionnement Général de l’Application en Ligne	16
9.2.1	Contexte et Choix de la Solution de Déploiement	16
9.2.2	Principe de Fonctionnement de Ngrok	17
9.2.3	Tests Fonctionnels	17
10	Conclusion	19

1 Introduction

Ce document a pour objectif de définir, organiser et décrire l'ensemble des tests réalisés pour vérifier que le logiciel respecte intégralement le cahier des charges. Il s'agit d'un outil essentiel à la validation du produit final, garantissant que toutes les fonctionnalités – telles que définies par la maîtrise d'ouvrage – sont correctement implémentées, ainsi que les cas d'utilisation ou contraintes supplémentaires envisagées par la maîtrise d'œuvre.

Les tests fonctionnels garantissent que l'application répond aux attentes du client et couvrent les fonctionnalités minimales décrites dans le cahier de recette.

Les tests d'intégration vérifient le bon fonctionnement des interactions entre les différents modules (extraction des caractéristiques, indexation via FAISS, interface utilisateur, etc.). Les tests unitaires se concentrent sur la validation isolée de chaque module ou classe, en s'appuyant sur des jeux d'essai prédéfinis (données d'entrée, résultats attendus, critères d'évaluation).

Ce plan de tests va donc au-delà du cahier de recette en couvrant des cas d'utilisation complémentaires et des contraintes techniques non abordées initialement dans le cahier des charges.

À noter que ce plan de tests est sujet à évolution en fonction des découvertes des développeurs et changement de plan. Ainsi, plusieurs versions peuvent et seront rendues au fur et à mesure du développement. La version du document est indiquée à la deuxième page de celui-ci, et dans le descriptif sur La Forge.

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.
 - **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 appliquée 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 plan de tests vise à aider les développeurs, soit la maîtrise d'œuvre, à assurer une couverture complète des fonctionnalités de l'application, valider au plus près l'intégration des modules, et garantir une expérience utilisateur conforme aux objectifs du projet. Chaque test doit être croisé avec le cahier des charges et le cahier de recette pour s'assurer que toutes les exigences fonctionnelles et techniques sont bien prises en compte.

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:

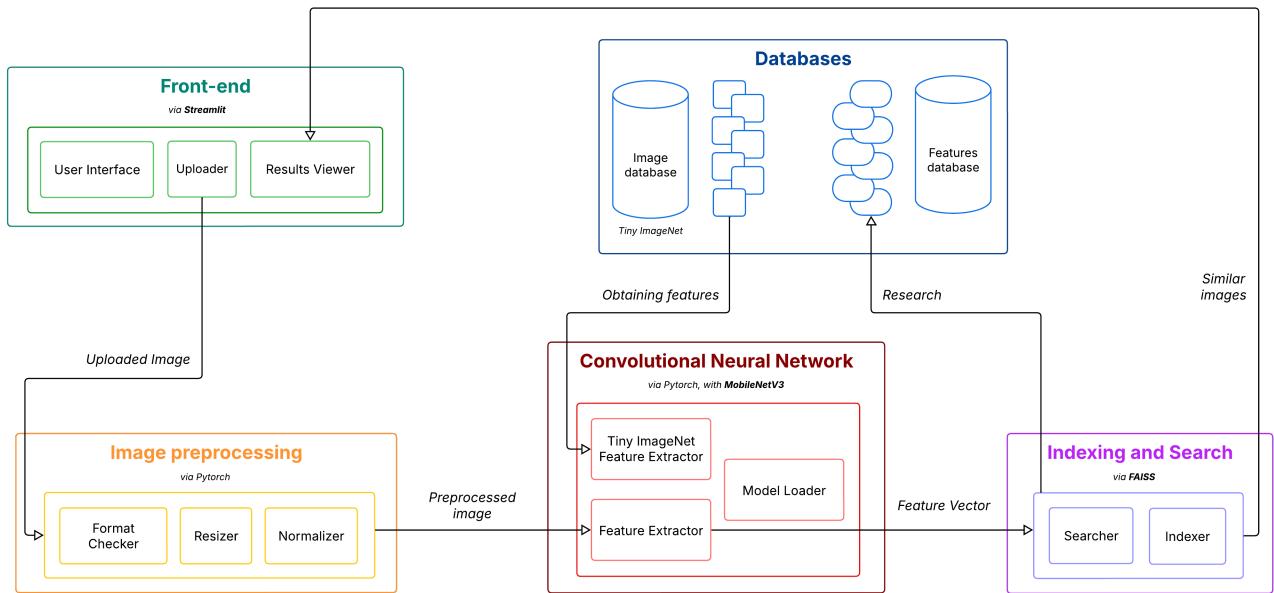


Figure 1: Schéma de l'architecture globale du système

6 Organisation et Ordonnancement des Tests

6.1 Stratégie globale

L'ordonnancement des tests est établi de façon à commencer par les tests unitaires, afin d'assurer la qualité de chaque module développé. Puis, enchaîner avec les tests d'intégration pour vérifier les interfaces entre les modules et valider les cas d'usage principaux. Enfin, terminer par une série complète de tests fonctionnels (complémentaires à ceux du cahier de recette) afin de couvrir l'ensemble des fonctionnalités, y compris les cas d'utilisation particuliers et les contraintes techniques supplémentaires (par exemple, tests de performance et de gestion d'erreurs).

- Planification et analyse des besoins : Une phase initiale où toutes les exigences fonctionnelles et techniques du projet sont définies de manière détaillée.
- Conception : Une phase dédiée à la conception de l'architecture du système, y compris la structure de la base de données et les interfaces utilisateur.
- Implémentation : Le développement du système est réalisé selon les spécifications définies, en respectant l'architecture conçue.
- Tests et validation : Une fois le développement terminé, des tests unitaires, d'intégration et fonctionnels sont effectués pour garantir la conformité aux exigences initiales.
- Déploiement et maintenance : Après validation, le système est déployé et des mises à jour correctives ou évolutives peuvent être apportées en fonction des besoins.

6.2 Environnement de test

Les tests seront réalisés sur différents environnements pour simuler les conditions réelles d'utilisation, incluant :

- Plusieurs systèmes d'exploitation (Windows, Linux, macOS)
- Différents navigateurs web pour l'interface (Chrome, Firefox, Edge, Safari)
- Une base de données optimisée pour le stockage et la recherche via FAISS

7 Description des Tests Unitaires

Les tests unitaires sont réalisés pour chacun des modules et classes développés. Pour chaque module, plusieurs jeux d'essai seront utilisés afin de couvrir l'ensemble des scénarios possibles.

PyTest sera utilisé : ce framework de test pour Python est utilisé pour automatiser les tests unitaires et s'assurer que chaque composant fonctionne comme prévu.

Unittest sera aussi utilisé : c'est le module intégré de Python pour l'écriture de tests unitaires.

7.1 Module 1 – Prétraitement des Images

- **Mise en œuvre**

Vérification des fonctions de redimensionnement, de normalisation, de conversion, vérification de format, vérification d'existence.

- **Jeux d'essai**

- **Jeu 1**

Données d'entrée : image de format JPEG/JPG en haute résolution.

Résultat attendu : image redimensionnée selon la taille cible (ex. 224×224), correctement normalisée.

Critère d'évaluation : comparaison pixel par pixel.

- **Jeu 2**

Données d'entrée : image PNG avec canaux alpha.

Résultat attendu : retrait du canal alpha et conversion avec prétraitement conforme.

Critère d'évaluation : vérification de la transformation et affichage correct dans le module suivant.

- **Jeu 3**

Données d'entrée : image de format interdit tel que WEBP/GIF/PDF/...

Résultat attendu : message d'erreur lié au format interdit.

Critère d'évaluation : message d'erreur correct, l'application reste opérationnelle et ne rencontre pas de crash.

- **Jeu 4**

Données d'entrée : image de format JPEG/JPG/PNG avec l'option de conversion en tenseur activée lors de l'appel du module.

Résultat attendu : un tenseur aux standards d'ImageNet.

Critère d'évaluation : le tenseur est reconnu par les autres modules.

- **Jeu 5**

Données d'entrée : un chemin d'image erroné

Résultat attendu : message d'erreur lié à l'image introuvable.

Critère d'évaluation : message d'erreur correct, l'application reste opérationnelle et ne rencontre pas de crash.

7.2 Module 2 – Extraction des Caractéristiques (CNN)

- **Mise en œuvre**

Validation de l'implémentation du modèle CNN (MobileNet) avec chargement des poids pré-entraînés et extraction des deep features.

- **Jeux d'essai**

- **Jeu 1**

Données d'entrée : aucune, on teste ici uniquement le chargement du CNN.

Résultat attendu : chargement correct du CNN.

Critère d'évaluation : temps de chargement.

- **Jeu 2**

Données d'entrée : image correctement prétraitée.

Résultat attendu : vecteur de caractéristiques d'une dimension conforme (ex. 1024 éléments).

Critère d'évaluation : concordance de la dimension, valeurs comprises dans un intervalle attendu.

- **Jeu 3**

Données d'entrée : image partiellement altérée (bruit ajouté).

Résultat attendu : vecteur de caractéristiques généré et robustesse de l'extraction.

Critère d'évaluation : stabilité du vecteur comparé à une version nettoyée (écart maximum toléré).

- **Jeu 4**

Données d'entrée : aucune, on teste ici que le modèle est transféré sur le bon appareil (CPU/GPU).

Résultat attendu : transfert sur le GPU s'il y a, sinon sur le CPU.

Critère d'évaluation : transfert correct.

7.3 Module 3 – Indexation et Recherche via FAISS

- **Mise en œuvre**

Vérification de la création et gestion de l'index FAISS, ainsi que de la recherche approximative des plus proches voisins.

- **Jeux d'essai**

- **Jeu 1**

Données d'entrée : jeu de vecteurs simulant une base d'images.

Résultat attendu : indexation complète et rapide, recherche renvoyant les vecteurs les plus pertinents.

Critère d'évaluation : temps de traitement, taux de correspondance correct selon une métrique (par exemple, précision ou score de similarité).

- **Jeu 2**

Données d'entrée : requête avec vecteur erroné ou non conforme.

Résultat attendu : gestion d'erreur appropriée, message d'erreur et non-exécution du traitement erroné.

Critère d'évaluation : vérification des conditions d'erreur et robustesse de la

fonction.

Ce module contient aussi des fonctions de recherches des chemins d'images, et de recherche des catégories similaires. Ainsi les tests les concernant sont les suivants :

– **Jeu 3 : recherche des catégories similaires**

Données d'entrée : vecteur simulant une image.

Résultat attendu : la liste des catégories, de bonne longueur, avec les distances liées aux catégories.

Critère d'évaluation : l'existence des catégories, la bonne valeur de distance.

– **Jeu 4 : recherche des chemins d'images**

Données d'entrée : un index arbitraire.

Résultat attendu : le chemin de image lié à l'index.

Critère d'évaluation : rapidité et chemin d'image valide.

7.4 Module 4 – Interface Utilisateur et Communication Back-end

- **Mise en œuvre**

Tester les composants de l'interface (formulaire d'upload, affichage des résultats) et leur communication avec le backend via API.

- **Jeux d'essai**

– **Jeu 1**

Données d'entrée : image chargée par l'utilisateur via le formulaire.

Résultat attendu : envoi correct et affichage cohérent des résultats.

Critère d'évaluation : temps de chargement, conformité du rendu graphique.

– **Jeu 2**

Données d'entrée : simulation d'une connexion lente ou interruption réseau.

Résultat attendu : l'interface reste stable, un message d'erreur informatif s'affiche.

Critère d'évaluation : robustesse de l'interface et gestion des erreurs côté client.

7.5 Module 5 – Recherche de similarité via CLIP

- **Rappel**

En guise de fonctionnalité optionnelle, nous avons implémenté une technologie plus récente appelée Text-Based Image Retrieval (TBIR). Contrairement à la CBIR, qui compare les images par similarité visuelle, la TBIR permet aux utilisateurs d'effectuer des recherches en langage naturel, par exemple en saisissant "un chien dans un jardin" pour récupérer des images visuellement pertinentes.

Pour ce faire, nous avons utilisé CLIP, un modèle développé par OpenAI. CLIP signifie "Contrastive Language–Image Pretraining" (pré-entraînement en langage contrasté), et a été entraîné sur des centaines de millions de paires image-texte provenant d'Internet. L'idée principale est que CLIP peut intégrer images et texte

dans le même espace vectoriel, afin de les comparer directement grâce à des mesures de similarité.

Comme avec MobileNet dans notre pipeline CBIR, nous extrayons les vecteurs de caractéristiques pour toutes les images du jeu de données, mais cette fois en utilisant le CNN de CLIP. Lorsqu'une invite de texte est saisie, elle est encodée par le modèle de texte de CLIP et les vecteurs d'images les plus similaires sont récupérés. Cette combinaison permet à CLIP de comprendre et de faire correspondre les concepts visuels et textuels, ce qui en fait un outil puissant qui comble le fossé entre la vision par ordinateur et le langage naturel.

- **Mise en œuvre**

Vérifier que le modèle CLIP est correctement chargé, que le texte est bien transformé en vecteur, et que la similarité avec les images fonctionne.

- **Jeux d'essai**

- **Jeu 1**

Données d'entrée : requête textuelle simple (a dog, a red car, a fruit).

Résultat attendu : vecteur normalisé de la bonne dimension.

Critère d'évaluation : dimensions, norme, pas de valeurs NaN/infinités.

- **Jeu 2**

Données d'entrée : chaîne vide, texte avec caractères spéciaux uniquement.

Résultat attendu : levée d'exception ou gestion robuste de l'entrée.

Critère d'évaluation : pas de crash, message d'erreur clair si applicable.

- **Jeu 3**

Données d'entrée : a dog et a puppy .

Résultat attendu : résultats très similaires.

Critère d'évaluation : évaluation de la stabilité (intersection des résultats ou faible distance entre vecteurs résultats).

- **Jeu 4**

Données d'entrée : forcer l'utilisation de GPU ou CPU.

Résultat attendu : modèle chargé sur l'appareil correct selon la disponibilité.

Critère d'évaluation : cohérence avec la variable `device` et temps de traitement raisonnable.

8 Description des Tests d’Intégration

Pour chacun des tests d’intégration, la procédure est détaillée selon trois axes : description externe, but du test et principe de réalisation.

8.1 Test d’Intégration n°1 – Recherche d’image par similarité

- **Description du test**

L’utilisateur charge une image via l’interface web. Le système effectue le prétraitement, passe l’image dans le modèle CNN pour extraire les deep features, qui sont ensuite comparées aux vecteurs déjà indexés dans FAISS. Les images les plus proches (selon un score de similarité) sont renvoyées et affichées.

- **But du test**

Vérifier que l’ensemble du pipeline (upload → prétraitement → extraction → indexation → recherche) fonctionne correctement et renvoie des résultats pertinents en moins de 3 secondes. En d’autres termes, c’est s’assurer que tous les composants techniques communiquent correctement pour délivrer un résultat pertinent, performant et cohérent avec la requête utilisateur.

- **Principe de réalisation**

- Étape 1 : Lancer l’application dans l’environnement de test.
- Étape 2 : Utiliser un jeu d’images de test (images avec différents formats, niveaux de qualité, et sujets) pour simuler des requêtes utilisateurs.
- Étape 3 : Vérifier que pour chaque image soumise, les images similaires affichées sont pertinentes
- Paramètres : temps de réponse, taux de précision (nombre correct sur l’ensemble des résultats), gestion d’erreur en cas d’image non valide.

8.2 Test d’Intégration n°2 – Affichage et Interaction de l’Interface

- **Description du test**

Vérifier que les résultats de recherche sont affichés correctement sous le format d’une mosaïque et que l’interface permet une navigation intuitive.

- **But du test**

S’assurer de la cohérence et de l’ergonomie de l’interface utilisateur permettant une interaction fluide, et garantissant ainsi une bonne expérience utilisateur même en cas de forte sollicitation, et quels que soient le support et le navigateur utilisés.

- **Principe de réalisation**

- Étape 1 : Lancer l’application et effectuer une recherche avec une image type.
- Étape 2 : Observer le rendu graphique sur différents navigateurs et résolutions d’écran.
- Étape 3 : Tester les interactions (clic sur une image, changement de mode d’affichage, etc.) et relever tout dysfonctionnement éventuel.
- Paramètres : compatibilité cross-browser (ex: Chrome, Firefox, Edge) , temps de rafraîchissement de l’interface, réactivité aux interactions, réactivité au clic, comportement lors du redimensionnement de l’écran (responsive design), Accessibilité

(navigation clavier, alt des images)

8.3 Test d'Intégration n°3 – Gestion des Erreurs et Cas Particuliers

- **Description du test**

Simuler l'envoi d'images corrompues, de formats non pris en charge ou de requêtes mal formées (JSON invalide, par exemple) afin de vérifier la robustesse du système et la qualité des messages d'erreur renvoyés.

- **But du test**

Vérifier la robustesse et la capacité du système à gérer les entrées invalides sans crash, et à informer correctement l'utilisateur.

- **Principe de réalisation**

- Étape 1 : Envoyer différentes images erronées ou mal formées via l'interface d'upload.
- Étape 2 : Observer et noter les réponses : messages d'erreur explicites et non bloquants, codes HTTP appropriés (400, 500, etc.).
- Paramètres : type d'erreur, clarté du message, absence de crash ou de blocage du service.

9 Description des Tests Fonctionnels

Les tests fonctionnels ont pour but de valider le bon comportement du système selon les spécifications attendues du point de vue de l'utilisateur. Contrairement aux tests unitaires qui se concentrent sur des fonctions isolées, les tests fonctionnels évaluent le système dans son ensemble, en s'assurant que les différentes fonctionnalités, telles que la recherche d'image ou l'interprétation d'une requête textuelle, fournissent des résultats pertinents, cohérents et utilisables dans des conditions réalistes.

9.1 Test à Large Échelle – Évaluation Qualitative des Résultats

Cette section vise à évaluer de manière fonctionnelle la pertinence des résultats retournés par le système, tant pour la recherche par image (CBIR) que pour la recherche par texte (TBIR). Pour ce faire, un ensemble varié d'images et de requêtes textuelles a été utilisé afin de simuler des scénarios réalistes d'utilisation.

Deux métriques sont évaluées :

- La pertinence du **Top-1** résultat (la première image retournée).
- La pertinence globale du **Top-10**, mesurée comme le pourcentage d'images jugées pertinentes parmi les dix premières suggestions.

Les évaluations sont basées sur la similarité perçue par un utilisateur humain, en fonction du contexte de la requête (image ou texte).

9.1.1 Recherche par Image (CBIR)

- **Méthodologie :**

Un ensemble d'environ **200 images tests** a été sélectionné aléatoirement depuis différents domaines visuels : animaux, objets, véhicules, paysages, scènes urbaines, etc. Chaque image est soumise au système en tant que requête.

- **Résultats :**

- **Pertinence Top-1 :** 82% des premières images retournées sont jugées visuellement très proches de l'image de requête.
- **Pertinence Top-10 :** En moyenne, 76% des dix premières images sont jugées pertinentes.

- **Observations :**

Le système retourne des résultats très cohérents pour les images simples et bien définies. Par exemple, une image de montagne retourne sans difficulté d'autres paysages montagneux similaires, même avec des variations saisonnières ou d'angle de vue.

Cependant, des confusions peuvent apparaître entre certaines classes visuellement proches. Il n'est pas rare, par exemple, que le système retourne un tigre en réponse à une image de lion, ou qu'un léopard soit confondu avec un jaguar. Ce phénomène s'explique par la proximité visuelle entre ces espèces et le fait que le dataset Tiny ImageNet ne contient pas toutes les classes animales possibles, ce qui limite parfois la précision de correspondance.

9.1.2 Recherche par Texte (TBIR avec CLIP)

- **Méthodologie :**

Une série d'environ **100 requêtes textuelles** a été préparée, couvrant différents thèmes :

- Des objets simples (“a red apple”, “a yellow car”),
- Des scènes naturelles (“a snowy mountain”, “a beach at sunset”),
- Des animaux (“a brown bear”, “a small dog”),
- Des concepts plus généraux ou ambigus (“something dangerous”, “a cute creature”).

Pour chaque requête, la pertinence du premier résultat et la qualité globale du top-10 ont été évaluées selon la même méthodologie que pour le CBIR.

- **Résultats :**

- **Pertinence Top-1 :** **78,0%** des premières images sont jugées directement pertinentes visuellement.
- **Pertinence Top-10 :** En moyenne, **70,2%** des dix images retournées sont jugées cohérentes avec la requête textuelle.

- **Observations :**

Les requêtes précises comme “a red apple” ou “a mountain covered in snow” donnent des résultats de haute qualité, avec peu d'ambiguïtés visuelles. En revanche, des confusions apparaissent pour les concepts plus larges (“a cute animal” peut retourner aussi bien un chaton qu'un panda ou un objet n'ayant aucun rapport).

De plus, les erreurs peuvent survenir dans des cas de proximité visuelle forte ou de sémantique ambiguë. Par exemple, une requête “a lion” peut donner lieu à des images de tigres ou de grands félins similaires, surtout si la classe exacte n'existe pas dans le dataset d'embeddings.

Conclusion globale : Ces tests fonctionnels à grande échelle montrent une bonne robustesse du système dans des conditions réalistes. Bien que la précision soit très satisfaisante sur des objets ou scènes bien représentés dans le dataset, certaines limites apparaissent naturellement dans les cas de confusion visuelle ou de diversité limitée dans les classes indexées.

9.2 Test Fonctionnel – Fonctionnement Général de l'Application en Ligne

9.2.1 Contexte et Choix de la Solution de Déploiement

Pour tester l'application dans un environnement réel, un déploiement en ligne a été nécessaire. Cependant, du fait de la taille importante des ressources nécessaires au fonctionnement de l'application (dataset Tiny ImageNet, fichiers d'embeddings, fichiers de

catégories), il n'existe pas de solution gratuite ou simple permettant un hébergement complet de l'ensemble sur une plateforme cloud classique.

Afin de contourner cette limitation et de simuler un environnement d'accès à distance pour les tests, la solution retenue est l'utilisation de **Ngrok**.

Ngrok permet d'exposer un serveur local (exécuté en local sur la machine de développement) à Internet via un tunnel sécurisé. Cela permet aux testeurs et utilisateurs externes d'accéder à l'application comme s'il s'agissait d'un service déployé publiquement, tout en conservant les lourdes ressources en local.

9.2.2 Principe de Fonctionnement de Ngrok

Le fonctionnement est simple :

1. L'application est exécutée localement sur une machine.
2. Ngrok est lancé pour créer un tunnel vers le port d'écoute de l'application.
3. Un lien public HTTPS est généré par Ngrok (le même à chaque exécution), que l'on peut partager pour accéder à l'application à distance.

Ce système permet ainsi de tester toute l'application, y compris les étapes lourdes comme la recherche d'images ou la manipulation de vecteurs d'embeddings, sans avoir besoin de les transférer vers un cloud.

9.2.3 Tests Fonctionnels

Pour valider que l'application en ligne répond aux attentes du cahier des charges, plusieurs tests fonctionnels ont été définis :

- **Accessibilité à distance**
 - Tester l'accès depuis différents appareils et réseaux (ordinateur distant, smartphone, réseau universitaire, etc.).
 - Vérifier que la latence reste raisonnable pour les principales fonctionnalités.
- **Fonctionnement complet du pipeline**
 - Soumettre une requête via l'interface (image ou texte) et vérifier que la chaîne complète fonctionne :
 - * Upload de l'image ou saisie de texte.
 - * Prétraitement.
 - * Recherche via FAISS ou CLIP.
 - * Affichage des résultats.
 - Réaliser ce test plusieurs fois de suite pour s'assurer de la stabilité.
- **Affichage et Interactivité**
 - Vérifier que les résultats sont bien rendus côté front-end (images visibles, triables, navigation fluide).
 - Tester les fonctionnalités d'interaction : clic sur une image, chargement dynamique, redirection éventuelle, etc.

- **Robustesse et Résilience**

- Simuler des erreurs : requêtes invalides, images corrompues, texte vide, etc.
- Vérifier que l'application gère ces cas sans crash, avec messages d'erreur clairs.

- **Conformité au Cahier des Charges**

- Temps de réponse : vérifier que le délai entre l'envoi d'une requête et l'affichage des résultats reste sous les 5 secondes dans 90% des cas.
- Fiabilité : répéter les tests sur plusieurs jours et sessions pour détecter d'éventuelles instabilités.
- Pertinence des résultats : observer qualitativement les réponses retournées pour différentes requêtes.

10 Conclusion

Ce plan de tests a pour vocation de structurer et d'organiser l'ensemble des vérifications techniques et fonctionnelles afin de s'assurer que l'application développée répond parfaitement aux exigences définies, tant du point de vue de la maîtrise d'ouvrage que de la maîtrise d'œuvre.

L'exécution rigoureuse de ces tests, à la fois unitaires et d'intégration, garantira ainsi la qualité, la robustesse et la performance de votre produit final.

En cas de découverte de nouveaux cas d'utilisation ou de contraintes techniques pendant la phase de développement, le présent plan de tests devra être mis à jour afin d'en intégrer l'évaluation.

Toute anomalie relevée pendant l'exécution des tests devra être consignée et traitée, afin d'assurer la résolution rapide des problèmes identifiés.