

Classification de Chiens et de Chats

Avec transfer learning



CHUKWUEKE Jeremiah E.

BADAYODI Samson

Encdrant : M.BOUYO

Table des matières

[1.INTRODUCTION : 3](#_Toc170232275)

[2.ETAT DE L’ART : 3](#_Toc170232276)

[2.1 Introduction 3](#_Toc170232277)

[2.2 Développement d'API avec Flask 3](#_Toc170232278)

[2.3 Interface Utilisateur avec Streamlit 4](#_Toc170232279)

[2.4 Déploiement sur Google Cloud Platform (GCP) 4](#_Toc170232280)

[3. Méthodologie 4](#_Toc170232281)

[Développement Backend: 4](#_Toc170232282)

[Développement Frontend: 4](#_Toc170232283)

[Déploiement Cloud: 5](#_Toc170232284)

[4. Implémentation 5](#_Toc170232285)

[A. Partie Backend 5](#_Toc170232286)

[Code de l'API avec Flask 5](#_Toc170232287)

[B. Partie Frontend 6](#_Toc170232288)

[Code de l'Interface Utilisateur avec Streamlit 6](#_Toc170232289)

[C. Déploiement sur le Cloud 6](#_Toc170232290)

[ Construction et Déploiement de l'Image Docker: 6](#_Toc170232291)

[ Configuration de Google Cloud Platform: 6](#_Toc170232292)

[5. Limites ou Difficultés 6](#_Toc170232293)

[6. Conclusion 6](#_Toc170232294)

[7. Références 7](#_Toc170232295)

# 1.INTRODUCTION :

Le projet de classification d'images de chiens et de chats a pour objectif de développer un système capable de distinguer ces deux catégories d'animaux à partir d'images. Le projet se compose de trois grandes étapes: développement d'une API « backend », création d'une interface utilisateur « frontend » et déploiement de l'application sur le cloud. Ce rapport présente les étapes de la réalisation, les outils utilisés, les défis rencontrés et les résultats obtenus.

# 2.ETAT DE L’ART :

## 2.1 Introduction

Dans le cadre de ce projet, nous avons exploré différentes technologies et méthodes pour la création, l'intégration et le déploiement d'une API de classification d'images. Nous avons principalement utilisé Flask pour le développement de l'API, Streamlit pour la création de l'interface utilisateur et Google Cloud Platform (GCP) pour le déploiement. Cette section présente un aperçu des recherches et des pratiques actuelles dans ces domaines.

## 2.2 Développement d'API avec Flask

**Flask** est un framework en Python, reconnu pour sa simplicité et sa flexibilité. Il est largement utilisé pour développer des applications web légères et des API RESTful. Flask permet aux développeurs de créer rapidement des endpoints et de gérer les requêtes HTTP de manière efficace.

**Recherches** : Selon les travaux de Miguel Grinberg et les ressources disponibles sur le site officiel de Flask, la construction d'une API REST nécessite la mise en place de routes spécifiques pour les opérations CRUD (Create, Read, Update, Delete). Dans le contexte de la classification d'images, une route POST pour recevoir les images et retourner les prédictions est essentielle. L'utilisation de bibliothèques comme TensorFlow pour le traitement des images et la prédiction au sein de Flask est courante et bien documentée.

## 2.3 Interface Utilisateur avec Streamlit

**Streamlit** est un framework open-source en Python conçu pour créer rapidement des applications web interactives pour l'apprentissage automatique et la science des données. Il est apprécié pour sa simplicité d'utilisation et sa capacité à convertir des scripts Python en applications web interactives avec une quantité minimale de code.

**Recherches** : Selon les documents de Streamlit et les tutoriels disponibles en ligne, Streamlit permet de créer des interfaces utilisateur intuitives pour visualiser les résultats des modèles d'apprentissage automatique. Il offre des widgets interactifs tels que des boutons de téléchargement de fichiers, des boutons de commande, et des affichages d'images, ce qui le rend idéal pour notre application de classification d'images. Streamlit peut être facilement intégré avec des API backend développées en Flask, permettant une communication fluide entre le frontend et le backend.

## 2.4 Déploiement sur Google Cloud Platform (GCP)

**Google Cloud Platform (GCP)** offre une variété de services cloud permettant de déployer des applications de manière évolutive et fiable. Pour ce projet, nous avons exploré l'utilisation de Docker pour containeriser notre application et Google Cloud Run pour le déploiement.

**Recherches** : la containerisation avec Docker simplifie le processus de déploiement en assurant que l'application fonctionne de manière cohérente dans différents environnements. GCP propose des services comme Google Cloud Build pour automatiser la construction des conteneurs et Google Container Registry pour stocker les images Docker. Le déploiement Google Cloud Run permet de gérer facilement les versions.

# 3. Méthodologie

Le projet suit une approche en trois étapes:

## **Développement Backend:**

Utilisation de Flask pour créer une API REST capable de recevoir des images et de retourner des prédictions.

## **Développement Frontend:**

Utilisation de Streamlit pour créer une interface utilisateur permettant de télécharger des images et de visualiser les résultats de la classification.

## **Déploiement Cloud:**

Utilisation de Docker pour containeriser l'application et déploiement sur Google Cloud Platform (GCP).

# 4. Implémentation

## A. Partie Backend

### Code de l'API avec Flask

Initialisation de l'Application Flask:

* Nous importons les modules nécessaires et créons une instance de Flask (app).
* CORS est activé pour permettre les requêtes provenant de http://localhost:8501, où l'application Streamlit sera exécutée.
* Une instance de Api de flask\_restful est créée pour gérer les endpoints RESTful.

Définition de la Classe de Ressource:

* UploadImage est une classe héritant de Resource de flask\_restful.
* La méthode post est définie pour gérer les requêtes POST.

Récupération de l'Image:

* request.files.get("image") est utilisé pour obtenir l'image envoyée dans la requête.

Chargement et Compilation du Modèle:

* Le modèle cat\_dog.h5 est chargé et compilé avec adam comme optimiseur et binary\_crossentropy comme fonction de perte.

Prétraitement de l'Image:

* L'image est redimensionnée à 64x64 pixels et convertie en format RGB.
* Elle est ensuite convertie en un tableau numpy et ses dimensions sont étendues pour correspondre aux attentes du modèle.

Prédiction:

* Le modèle fait une prédiction sur l'image prétraitée.
* Si la prédiction est supérieure à un certain seuil (1), elle est classée comme "dog", sinon "cat".

Retour de la Réponse:

* Le résultat de la prédiction est renvoyé sous forme de JSON avec un code de statut HTTP 200 en cas de succès, ou 400 en cas d'erreur.

## B. Partie Frontend

### **Code de l'Interface Utilisateur avec Streamlit**

Description du Processus

1. Initialisation de l'Application Streamlit:
   * Titre de l'application et description de la fonctionnalité.
2. Téléchargement de l'Image:
   * Utilisation de st.file\_uploader pour permettre à l'utilisateur de télécharger une image au format JPG, JPEG ou PNG.
3. Affichage de l'Image:
   * L'image téléchargée est affichée dans la colonne 1 après redimensionnement à 200x200 pixels.
4. Classification de l'Image:

* Lorsque l'utilisateur clique sur le bouton "Classify", l'image est envoyée à l'API Flask via une requête POST.
* La réponse de l'API est affichée dans la colonne 2, montrant la prédiction (chien ou chat) et la confiance de la prédiction.

## C. Déploiement sur le Cloud

### Construction et Déploiement de l'Image Docker:

* Construire l'image Docker: docker build -t cat\_dog\_classifier .
* Déployer l'image sur Google Cloud Platform.

### Configuration de Google Cloud Platform:

* Créer un projet sur GCP.
* Configurer Google Cloud Run.
* déployer l'image Docker sur GCP.

# 5. Limites ou Difficultés

Lors de la réalisation de ce projet, un défi en particulier a été rencontré:

* **Configuration du Cloud:** La configuration et le déploiement sur GCP nécessitent une compréhension approfondie des services cloud et des conteneurs Docker.

# 6. Conclusion

Le projet de classification d'images de chiens et de chats a permis de mettre en œuvre une solution complète de bout en bout, comprenant une API backend, une interface frontend et un déploiement sur le cloud. Les résultats sont prometteurs, bien que certaines améliorations soient nécessaires pour améliorer la précision et l'efficacité de la solution.

# 7. Références

* Documentation Flask: https://flask.palletsprojects.com/
* Documentation Streamlit: https://docs.streamlit.io/
* Google Cloud Platform: <https://cloud.google.com/>
* Tutoriels et articles divers sur Medium, Stack Overflow et d'autres plateformes spécialisées.
* https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask