# Projet DS51

## Introduction

Ce rapport a pour but de détailler notre projet réalisé à l’occasion de l’UV DS51. Avant de parcourir les différentes étapes du projet, il nous semble adéquat de reformuler le sujet selon notre compréhension de celui-ci :

Notre but dans ce projet, est de, depuis une base d’images à récupérer (ImagesNet), organiser les différents groupes d’animaux au sein d’une ontologie RDF, puis d’appliquer un modèle de Machine Learning à chaque « nœud » de cette ontologie pour prédire la classification dans un des différents groupes d’une nouvelle image.

## Récupération des data

La banque d’images que nous devons récupérer est présente sur le site suivant : https://www.kaggle.com/competitions/imagenet-object-localization-challenge/data

Dans cette banque d’images (représentant 170 GB), sont classées plusieurs catégories d’entités, représentés par des ID. Toutes les photos représentant un type d’entité en particulier sont regroupées dans un dossier qui porte comme titre l’ID du type d’entité en question.

Une image contenant texte, capture d’écran, logiciel, Site web

Description générée automatiquement

Une image contenant texte, capture d’écran, nombre, diagramme

Description générée automatiquement

### Étape 1 : déchiffrer les ID

Bien que l’organisation des types d’entités soit fonctionnelle dans le projet ImagesNet, ces ID ne veulent pas dire grand-chose et nous allons devoir les déchiffrer. Pour cela, nous avons la chance de disposer du fichier « LOC\_synset\_mapping.txt », qui assimile à chaque ID plusieurs labels qui sont des synonymes. (Example : « great grey owl », « great gray owl », « Strix nebulosa »). Ainsi à l’aide d’une fonction simple, nous pouvons construire un dictionnaire pour récupérer chacun des libellés associés à l’ID de ImagesNet (nous nous serviront de ce dictionnaire plus tard).

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

***Figure 1 : contenu du fichier LOC\_synset\_mapping.txt***

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

***Figure 2 : fonction pour lire les données du fichier LOC\_synset\_mapping.txt***

### Étape 2 : Rechercher les entités dans WikiData

Maintenant que nous avons les noms des différentes entités, il va falloir faire une recherche sur WikiData pour récupérer les informations de cette entité. Cela est possible grâce à l’API de WikiData et l’utilisation de requête pour récupérer les informations nécessaires :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

***Figure 3 : fonction pour effectuer une recherche d’entité sur WikiData***

### Étape 3 : Repérer uniquement les dossiers d’animaux

Le but de ce projet est de construire une ontologie d’animaux, ainsi nous aimerions disposer uniquement de différents types d’animaux. Or, la base ImagesNet contient d’autres entités (objets, services…). Nous n’allons pas manuellement vérifier si chaque dossier contient ou non des animaux, ainsi nous allons utiliser le site <https://www.wikidata.org> pour nous aider dans cette tâche.

Pour savoir si une entité présente sur WikiData est un animal, on doit vérifier si l’entité est présent dans le « taxon » de WikiData, et est un sous-taxon de « animal » ( Q729 )

Ainsi nous allons développer une fonction, utilisant l’API de WikiData, pour détecter si une entité est sous-taxon de « animal» :

Une image contenant texte, capture d’écran, Police, document

Description générée automatiquement

***Figure 4 : fonction pour confirmer si une entité est un sous-taxon de « animal »***

### Étape 4 : Isoler les dossiers concernant des animaux

Maintenant que nous pouvons lire le fichier qui associe chaque id à un libellé et que nous pouvons, grâce à l’API de WikiData, rechercher une entité puis confirmer si oui ou non un libellé correspond à un animal, nous pouvons maintenant repérer uniquement les dossiers qui contiennent des animaux pour fabriquer notre ontologie. L’utilisation de toutes les fonctions précédentes nous donne ce fichier, via le format :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementUne image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

***Figure 5 : fichier contenant les ID WikiData, ImagesNet et la hiérarchie du taxon des animaux***

### Étape 5 : Data cleaning

Nous avons remarqué que dans WikiData, des éléments qui ne semblent pas être des animaux comme par exemple « diaper » (= « couche »), ou encore « knot » (= « nouer ») sont néanmoins reliés à la classe animal représentée par l’identifiant « Q729 », et ce via un chemin complexe qui représente une hiérarchie de parfois une centaine de couches différentes. Ainsi pour limiter les « faux animaux » trouvés grâce à la méthode du taxon, nous avons choisi de limiter les hiérarchies à une trentaine de couches, ce qui a également pour avantage de limiter notre nombre de types d’animaux.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

***Figure 6 : fonction qui supprime les « animaux » contenant une hiérarchie de plus de 30 couches***

A cause de la méthode utilisée pour récupérer les informations des entités depuis WikiData, nous avons des doublons. En effet, nous avons effectué la recherche sur toutes les appellations possibles d’une entité, nous nous retrouvons donc avec la même entité portant un nom différent dans notre fichier texte. *Exemple : Dans la figure 5, on constate que les deux premières lignes (tench) et (Tinca tinca) ont le même ID : « Q76280 », ce sont deux noms différents que porte le même animal.* Voici comment régler ce problème et supprimer les doublons :

***Une image contenant texte, capture d’écran, Police

Description générée automatiquement***

***Figure 7 : fonction qui supprime les doublons (mêmes animaux avec des noms différents)***

## Construction de l’ontologie

A la suite de l’étape de récupération de data précédente, nous avons un fichier texte contenant 215 animaux et leur hiérarchie (comment ils sont reliés à la classe « animal »). Une dernière étape est de connaître les noms des classes issues de la hiérarchie, ainsi notre hiérarchie sera composée de « mammifère », « oiseau » etc… plutôt que Q15473, Q71623 etc… :

Une image contenant texte, capture d’écran

Description générée automatiquement

***Figure 8 : fonction qui récupère le nom de l’ensemble de la hiérarchie et les note dans un fichier texte***

Nous allons maintenant constituer une ontologie pour classifier les différents types d’animaux, en utilisant la représentation RDF. Pour cela, nous allons utiliser une bibliothèque Python pour la réalisation d’ontologies RDF, rdflib.

Nous allons automatiser la création de code simple :

variable = URIRef('https://www.wikidata.org/wiki/[ID WikiData]')

g.add((variable, RDF.type, classe))

g.add((classe, RDF.type, sur-classe))

g.add((sur-classe, RDF.type, sur-sur-classe)) etc…

Ces lignes de code nous serviront à user de variables pour parler des entités et des classes, et à former la hiérarchie de classe de chaque entité jusqu’à la classe « animal ».

### Étape 1 : Générer le code de définition des variables

Nous allons générer le code automatiquement pour la définition des variables comme suit : Rappel : Le code à générer est : variable = URIRef('https://www.wikidata.org/wiki/[ID WikiData]')

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

***Figure 9 : fonction qui récupère le nom et l’ID de chaque animal, modifie le nom pour qu’il soit codable, et génère le code RDF***

### Étape 2 : Générer le code de définition de la hiérarchie

Pour définir la hiérarchie qui lie chaque animal à la classe « animal », nous allons utiliser toujours notre même fichier texte qui contient les animaux et leur hiérarchie sous forme d’identifiants, ainsi que notre fichier texte au préalablement créé qui associe chaque ID de la hiérarchie au nom de la classe.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

***Figure 10 : fonction qui génère le code RDF «g.add(( [animal], RDF.type, [type])) »***