PROJET DS51

Nicolas LECAS – Rémi BIZET - Samba CAMARA – Samy CHOUIT – Ziyi HUANG

UTBM INFO04 – Data Science

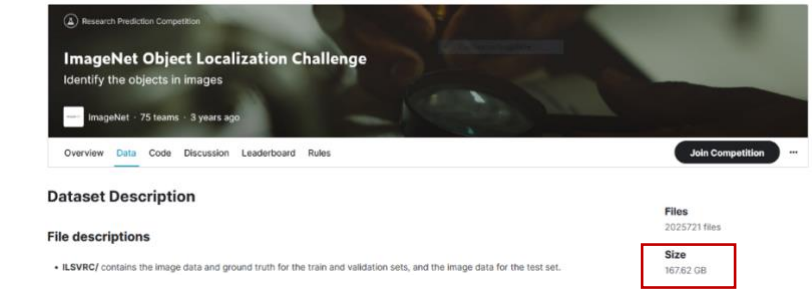
Table des matières

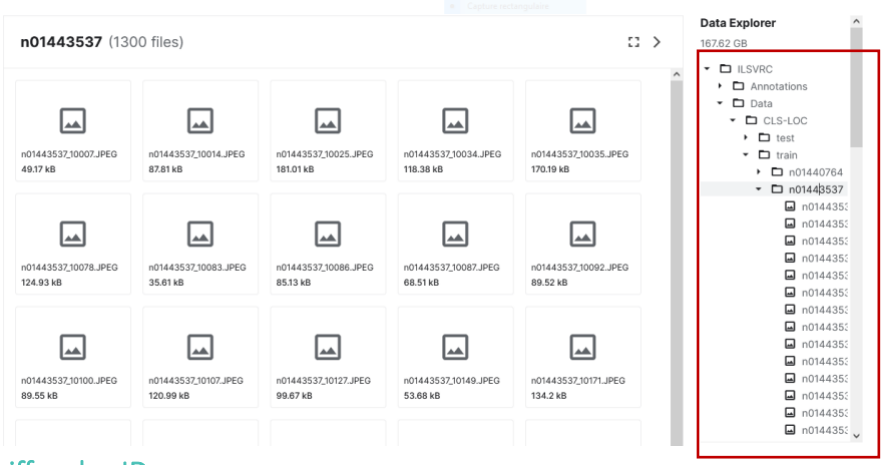
Introduction

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.





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. (Exemple : « 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).

LOC_synset_mapping.txt

```
n01440764 tench, Tinca tinca
n01443537 goldfish, Carassius auratus
n01484850 great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias
n01491361 tiger shark, Galeocerdo cuvieri
n01494475 hammerhead, hammerhead shark
n01496331 electric ray, crampfish, numbfish, torpedo
n01498041 stingray
n01514668 cock
n01514859 hen
n01518878 ostrich, Struthio camelus
n01530575 brambling, Fringilla montifringilla
n01531178 goldfinch, Carduelis carduelis
n01532829 house finch, linnet, Carpodacus mexicanus
n01534433 junco, snowbird
n01537544 indigo bunting, indigo finch, indigo bird, Passerina cyanea
n01558993 robin, American robin, Turdus migratorius
n01560419 bulbul
n01580077 jay
n01582220 magpie
n01592084 chickadee
n01601694 water ouzel, dipper
n01608432 kite
n01614925 bald eagle, American eagle, Haliaeetus leucocephalus
n01616318 vulture
n01622779 great grey owl, great gray owl, Strix nebulosa
n01629819 European fire salamander, Salamandra salamandra
n01630670 common newt, Triturus vulgaris
n01631663 eft
n01632458 spotted salamander, Ambystoma maculatum
n01632777 axolotl, mud puppy, Ambystoma mexicanum
```

Figure 1 : contenu du fichier LOC_synset_mapping.txt

```python
# Cette fonction construit un dictionnaire avec les différents labels inscrits dans le fichier LOC_synset_mapping.txt
def lire_fichier_texte(nom_fichier):
    dictionnaire_resultats = {}

    with open(nom_fichier, 'r') as fichier:
        lignes = fichier.readlines()

        for ligne in lignes:
            # Diviser la ligne en éléments séparés par une virgule
            elements = ligne.strip().split(',')

            # Récupérer les 9 premiers caractères comme valeur
            valeur = elements[0][:9]

            # Ajouter le premier élément comme clé avec la valeur associée
            dictionnaire_resultats[elements[0][10:].strip()] = valeur

            # Ajouter les éléments suivants comme clés avec la valeur associée
            for element in elements[1:]:
                dictionnaire_resultats[element.strip()] = valeur

    return dictionnaire_resultats
```

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 effectuer 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 :

```python
# Cette fonction renvoie un tableau de data correspondant à chaque élément cohérent à notre recherche
def recherche_entity_wikidata(nom_animal):
    # URL de l'API Wikidata
    url = 'https://www.wikidata.org/w/api.php'

    # Paramètres de la requête
    params = {
        'action': 'wbsearchentities',
        'format': 'json',
        'language': 'en',
        'type': 'item',
        'search': nom_animal
    }

    try:
        # Envoi de la requête à l'API
        response = requests.get(url, params=params)
        response.raise_for_status()

        # Analyse de la réponse JSON
        data = response.json()

        return data['search']

    except requests.exceptions.RequestException as e:
        print('Une erreur s\'est produite lors de la requête :', e)
```

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 présente 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» :

```python
# Cette fonction détermine si notre entité est une sous-catégorie de la classe "animal" dans le taxon
def is_subtaxon_of_animal(entity_id):
    # URL de l'API Wikidata pour récupérer les informations sur une classe
    url = 'https://www.wikidata.org/w/api.php'

    # Paramètres de la requête
    params = {
        'action': 'wbgetentities',
        'format': 'json',
        'ids': entity_id,
        'props': 'claims',
    }

    try:
        # Envoi de la requête à l'API
        response = requests.get(url, params=params)
        response.raise_for_status()

        # Analyse de la réponse JSON
        data = response.json()

        # Récupération de la classe parente
        if 'P171' in data['entities'][entity_id]['claims']:
            subtaxon_of_claims = data['entities'][entity_id]['claims']['P171']
            for claim in subtaxon_of_claims:
                if 'mainsnak' in claim and 'datavalue' in claim['mainsnak']:
                    datavalue = claim['mainsnak']['datavalue']
                    if 'value' in datavalue and 'id' in datavalue['value']:
                        parent_taxon_id = datavalue['value']['id']
                        if parent_taxon_id == 'Q729':
                            return True
                        else:
                            if is_subtaxon_of_animal(parent_taxon_id):
                                return True
                            else:
                                return False
        return False

    except requests.exceptions.RequestException as e:
        print('Une erreur s\'est produite lors de la requête :', e)
```

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

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 :

$$(nom\ de\ l'entité)\ \ [identifiant\ WikiData]\ [identifiant\ ImagesNet]\ taxon\ \ [hiérarchie]$$

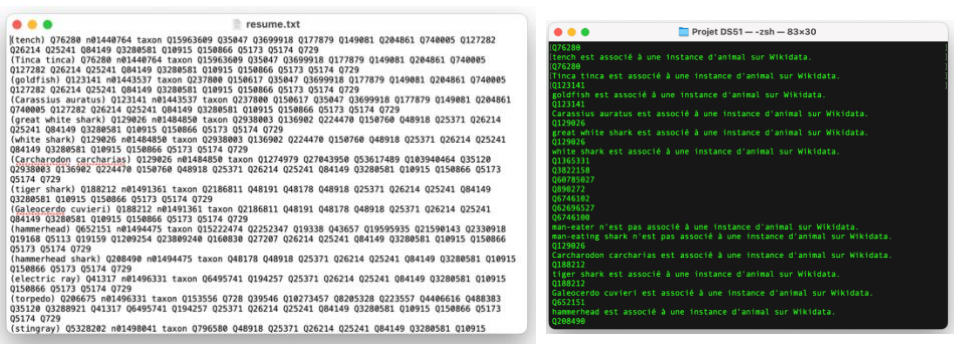


Figure 5 : fichier contenant les ID WikiData, ImagesNet et la hiérarchie du taxon des animaux

Étape 4 : 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.

```
# Ouvrir le fichier d'entrée en lecture
with open('resume.txt', 'r') as file:
    # Lire toutes les lignes du fichier
    lines = file.readlines()

# Créer une liste pour stocker les lignes à conserver
processed_lines = []

# Parcourir chaque ligne du fichier
for line in lines:
    # Séparer la ligne en entités
    entities = line.strip().split(' ')

    # Vérifier si la ligne contient moins ou égal à 4 entités
    if len(entities) <= 30:
        # Ajouter la ligne à la liste des lignes à conserver
        processed_lines.append(line)

# Ouvrir le fichier de sortie en écriture
with open('resume_filtre_30.txt', 'w') as file:
    # Écrire les lignes à conserver dans le fichier de sortie
    for line in processed_lines:
        file.write(line)

print("Lignes traitées ont été écrites dans le nouveau_fichier.txt.")
```

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 :

```
fichier_entree = open("resume_filtre_30.txt", "r")
fichier_sortie = open("resume_filtre_30_nodoublons.txt", "w")

identifiants_deja_rencontres = set()

for ligne in fichier_entree:
    fermeture_parenthese_index = ligne.find(")")
    if fermeture_parenthese_index != -1:
        texte_apres_parenthese = ligne[fermeture_parenthese_index + 1:].strip()
        mots_apres_parenthese = texte_apres_parenthese.split()
        if len(mots_apres_parenthese) > 0:
            identifiant = mots_apres_parenthese[0] # Le premier "mot" après la parenthèse ")"
            if identifiant not in identifiants_deja_rencontres: # Si l'identifiant est unique
                fichier_sortie.write(ligne)  # Écriture de la ligne dans le fichier de sortie
                identifiants_deja_rencontres.add(identifiant)

fichier_entree.close()
fichier_sortie.close()
```

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 »). Pour ce projet, nous devons seulement garder 6 animaux, (loup, requin, aigle, ours, passerin indigo, poisson rouge). Nous gardons donc dans le fichier texte uniquement les animaux qui nous intéressent. Maintenant, nous devons connaître les noms des classes issues de la hiérarchie, (ce qui relie notre animal à la classe « Animal »), ainsi notre hiérarchie sera composée de « mammifère », « oiseau » etc… plutôt que Q15473, Q71623 etc… :

```python
def get_entity_name(entity_id):
    # Construction de l'URL de requête à l'API Wikidata
    url = f"https://www.wikidata.org/w/api.php?action=wbgetentities&ids={entity_id}&format=json"

    # Envoi de la requête à l'API Wikidata
    response = requests.get(url)
    data = response.json()

    # Récupération du nom de l'entité
    try:
        entity_name = data["entities"][entity_id]["labels"]["en"]["value"]
        return entity_name
    except KeyError:
        return None

def get_unique_words_after_taxon(file_path):
    unique_words = set()
    with open(file_path, 'r') as file:
        for line in file:
            words = line.strip().split()
            if 'taxon' in words:
                taxon_index = words.index('taxon')
                for word in words[taxon_index + 1:]:
                    if word not in unique_words:
                        unique_words.add(word)
    return list(unique_words)

maliste = get_unique_words_after_taxon("resume.txt")

i =0
for entity_id in maliste:

    entity_name = get_entity_name(entity_id)
    if entity_name:
        with open("hierarchy.txt", "a") as fichier:
            hierarchy = " ".join(str(valeur) for valeur in tab_of_hierarchy)
            fichier.write(entity_name + " " + entity_id + "\n")
        i +=1
        print(i)
    else:
        print(f"L'entité {entity_id} n'a pas été trouvée.")
```

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

Cependant, le taxon WikiData est complexe, et les animaux sont séparés de la classe-mère « Animal » par de multiples couches (environ une quinzaine). *Exemple : requin est séparé de « animal » par 14 sous-classes différentes.* Ainsi nous aurions pu garder la méthode précédente si nous avions mené une étude sur toutes les espèces d'animaux trouvées au préalable, mais étant donné que nous n'avons gardé que 6 animaux, par la même occasion nous allons réduire notre « arbre hiérarchique » en choisissant les sous-classes qui séparent nos animaux de « Animal » et en en supprimant la plupart. Ce choix d'implémentation nous mène à l'arbre hiérarchique et l'arbre de caractéristiques suivants :

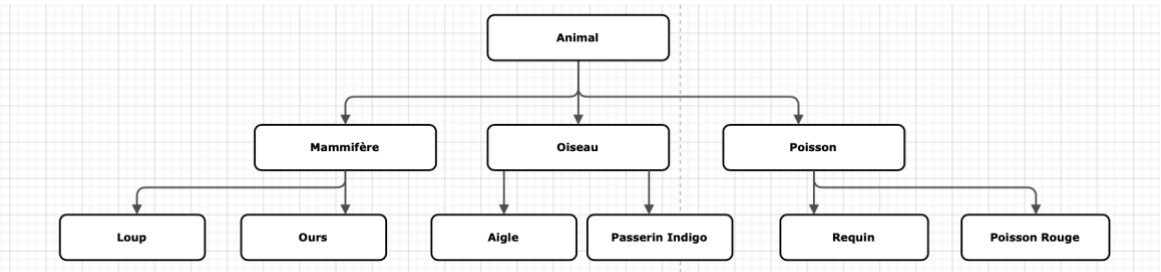


Figure 9 : Arbre hiérarchique des animaux

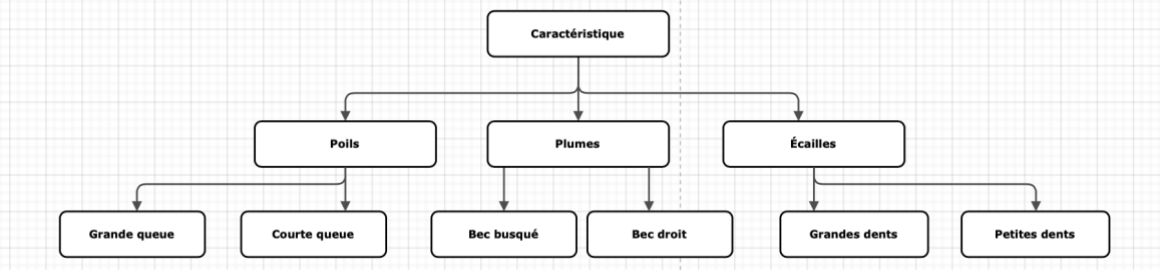


Figure 10 : Arbre des caractéristiques

Nous reste à construire l'ontologie OWL. Pour se faire, nous allons utiliser Python et plus précisément la bibliothèque rdflib, qui permet d'initier des classes, construire des propriétés, axiomes, restrictions…
Pour cette étape, nous vous renvoyons au fichier « ontologie.ipynb », qui est un notebook détaillé qui construit l'ontologie suivante :

Définition des préfixes :

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

Déclaration du premier niveau des deux arbres, restrictions pour les nœuds de l'arbre « Animal » :

```
owl:Ontology a owl:Class .

<http://exemple.com/ontologies/animaux.owl#Animal> a owl:Class .

<http://exemple.com/ontologies/animaux.owl#Mammifère> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Animal> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom <http://exemple.com/ontologies/animaux.owl#Poils> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Oiseau> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Animal> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom <http://exemple.com/ontologies/animaux.owl#Plumes> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Poisson> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Animal> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom <http://exemple.com/ontologies/animaux.owl#Écailles> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .


<http://exemple.com/ontologies/animaux.owl#Caractéristique> a owl:Class .

<http://exemple.com/ontologies/animaux.owl#Plumes> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Caractéristique> .

<http://exemple.com/ontologies/animaux.owl#Poils> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Caractéristique> .

<http://exemple.com/ontologies/animaux.owl#Écailles> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Caractéristique> .
```

Déclaration de la propriété « Possède » :

```
<http://exemple.com/ontologies/animaux.owl#Possède> a owl:ObjectProperty ;
    rdfs:domain <http://exemple.com/ontologies/animaux.owl#Animal> ;
    rdfs:range <http://exemple.com/ontologies/animaux.owl#Caractéristique> .
```

Déclaration du second niveau des deux arbres restrictions pour les nœuds de l'arbre « Animal » :

```
<http://exemple.com/ontologies/animaux.owl#Aigle> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Oiseau> ;
                    rdf:rest [ a owl:Restriction ;
                           owl:allValuesFrom <http://exemple.com/ontologies/animaux.owl#BecBusqué> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Passerin_indigo> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Oiseau> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom <http://exemple.com/ontologies/animaux.owl#BecDroit> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Loup> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Animal> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom
<http://exemple.com/ontologies/animaux.owl#LongueQueue> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Ours> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Mammifère> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom
<http://exemple.com/ontologies/animaux.owl#CourteQueue> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Poisson_rouge> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Poisson> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom
<http://exemple.com/ontologies/animaux.owl#PetitesDents> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .

<http://exemple.com/ontologies/animaux.owl#Requin> owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf [ rdf:first <http://exemple.com/ontologies/animaux.owl#Poisson> ;
                    rdf:rest [ a owl:Restriction ;
                            owl:allValuesFrom
<http://exemple.com/ontologies/animaux.owl#GrandesDents> ;
                            owl:onProperty <http://exemple.com/ontologies/animaux.owl#Possède> ],
                        () ] ] .


<http://exemple.com/ontologies/animaux.owl#BecBusqué> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Plumes> .

<http://exemple.com/ontologies/animaux.owl#BecDroit> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Plumes> .

<http://exemple.com/ontologies/animaux.owl#CourteQueue> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Poils> .

<http://exemple.com/ontologies/animaux.owl#GrandesDents> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Écailles> .

<http://exemple.com/ontologies/animaux.owl#LongueQueue> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Poils> .

<http://exemple.com/ontologies/animaux.owl#PetitesDents> a owl:Class ;
    rdfs:subClassOf <http://exemple.com/ontologies/animaux.owl#Écailles> .
```

Figure 11 : Ontologie OWL

Les classes de l'arbre « Animal » sont « reliées » aux classes de l'arbre « Caractéristique ». En effet, chaque classe de l'arbre « Animal » contient une restriction sur la portée de la propriété « Possède ». Ainsi, un « Aigle » est un « Oiseau » (sous classe de la classe « Oiseau ») qui « Possède » un « BecBusqué » (restriction sur la portée de la propriété « Possède »).

Construction des modèles de prédiction

Maintenant que notre ontologie est construite, c'est l'heure de construire les modèles de Machine Learning pour la classification. Pour la labellisation d'une nouvelle image, chaque niveau de l'arbre correspondra à un algorithme de classification qui indiquera à l'image le « chemin à suivre » pour le niveau suivant.
Exemple : On veut classer une nouvelle image d'Animal :

- *Un premier algorithme déterminera s'il s'agit d'un Mammifère, un Oiseau ou un Poisson*
- *En fonction de cette première prédiction, un second algorithme déterminera quel type de Mammifère, Oiseau ou Poisson nous avons rencontré (Ours/Loup, Passerin/Aigle et Requin/Poisson rouge)*

Ainsi nous allons utiliser l'algorithme GoogleNet pour la classification d'image à chaque niveau de l'arbre. La construction du modèle se résume en ses quelques étapes, toutes détaillées dans les fichiers du dossier construct_model.

- Spécification des images en tant qu'images de training et de tests
- Lecture des images et labellisation
- Transformation des images en tenseur
- Entraînement du modele avec les données du "train"
- Faire des prédictions sur les données de train et de test pour vérifier si les pourcentages de prédictions sont bons

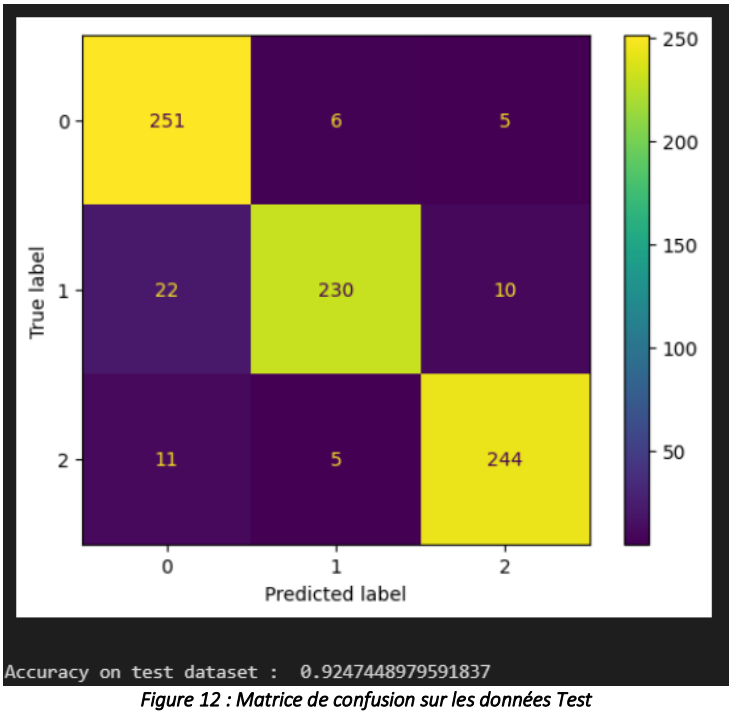


Figure 12 : Matrice de confusion sur les données Test

Cette matrice démontre l'efficacité de l'algorithme sur 1 des 4 modèles que nous avons implémentés. Il faut lire les performances de cette matrice en observant la diagonale. Plus la matrice s'approche d'une matrice identité, plus le modèle est précis. Ici, on peut voir dans la colonne de gauche par exemple que sur 284 images (251 + 22 + 11) de test qui devraient être classées « 0 », 251 le sont, 22 sont classées « 1 », et 11 sont classées « 2 ».

Implémentation de l'interface

Pour mettre en pratique notre programme, nous avons choisi d'implémenter une interface utilisateur simple mais fonctionnelle. Vous retrouverez le code de celle-ci dans le fichier gui.py, et une vidéo de démonstration : Demonstration.mkv. Après sélection de l'image à classifier par l'utilisateur, la logique du code est somme toute plutôt simple :

- Passage de l'image dans l'algorithme de classification entre "Mammifère", 'Oiseau" et "Poisson"
- En fonction de la classification précédente, passage de l'image dans l'algorithme de classification binaire entre "Loup" et "Ours" ou "Passerin indigo" et "Aigle" ou "Requin" et "Poisson rouge"

Le résultat de cette classification ainsi que le pourcentage de certitude d'appartenance à chaque classe est ensuite affiché sous forme de labels pour l'utilisateur, accompagné d'un diagramme qui indique la position de l'image dans l'arbre après classification.

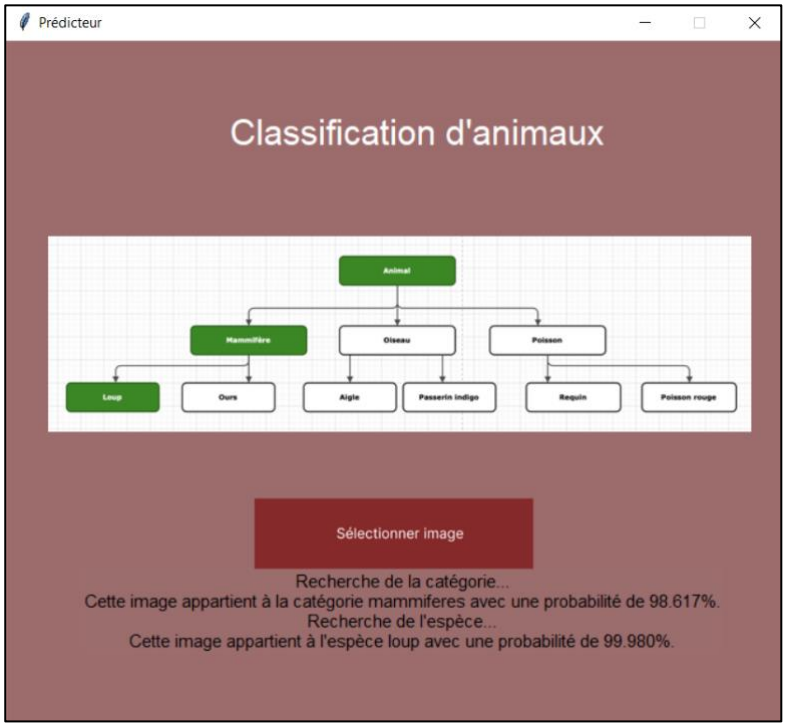


Figure 13 : Interface du projet

Conclusion

Ce projet nous a permis de nous familiariser avec les ontologies, comprendre leurs mécanismes (subClass, restrictions…) et leur utilité dans un but de classification pour construire un arbre hiérarchique. Nous avons aussi progressé dans la récupération d'image (data scraping), l'utilisation d'API pour récupérer des informations depuis WikiData et le Data Cleaning pour isoler les data intéressantes. Ces différentes compétences, couplées avec la construction de modèle de Machine Learning, ont fait de ce travail un projet complet.