

Saint-Upéry Gaspard

Lydia Renaud

Rapport de projet



IA et Big Data

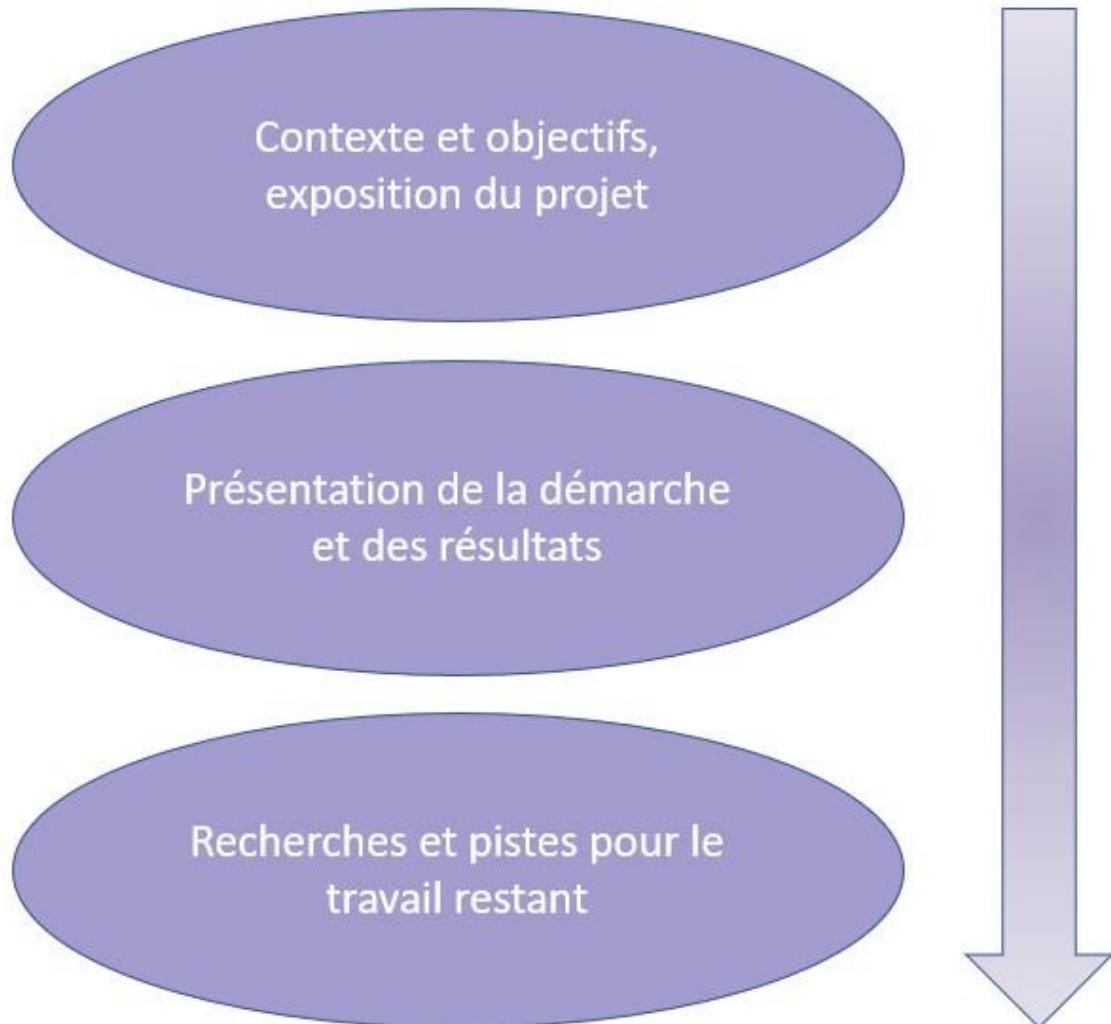
Détection de dépôt d'ordures illégaux dans la nature

Avec la participation de

Fabien Lahoudère

Matthias Spisser

Table des matières



Le projet IA et détection d'image que nous avons participé à développer entre octobre 2022 et qui se prolonge jusqu'en mars 2023 correspond à un partenariat avec Fabien Lahoudere et Mattias Spisser.

Ce projet vient se greffer sur une recherche plus vaste à laquelle participent les différents membres de l'équipe : concevoir un véhicule autonome qui détecte des dépôts d'ordures illégaux dans la nature.

C'est un projet orienté **big data et IA**.

Une caméra sera placée sur un véhicule autonome.



Ce véhicule autonome se déplacera en zone rurale, en acquisition constante.





Le but est d'entraîner un algorithme capable de détecter les dépôts d'ordures dans la nature et éventuellement de les classer selon leur type : sac poubelle, carton, déchet plastique, ...

La détection d'image n'est pas une simple reconnaissance d'image : elle consiste à détecter une image dans l'image du film de la caméra. Dans le film, les objets qu'on recherche peuvent être tournés dans des angles improbables voire éclairés par une lumière de tons inhabituels.

Pour gérer cette subtilité, on ne peut pas se contenter d'un simple algorithme KNN.

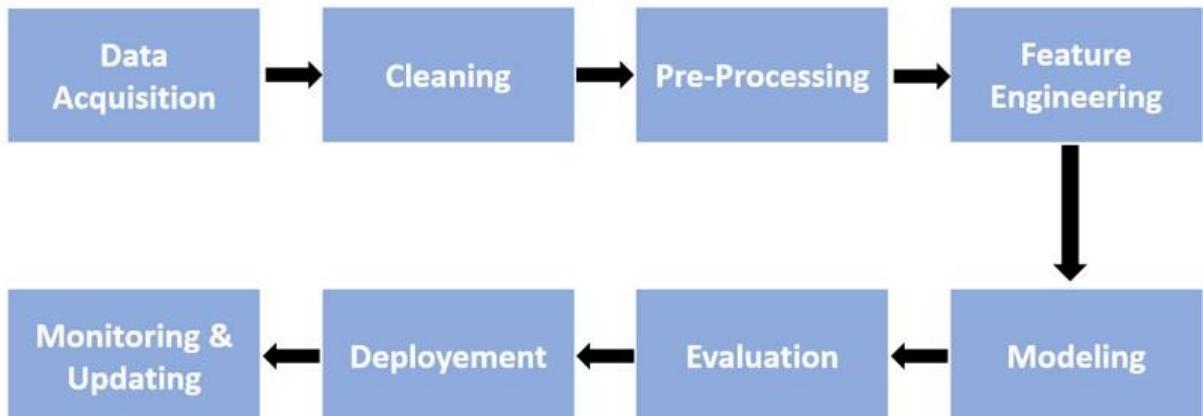
Certains classifieurs comme le SVM donnent de bons résultats pour reconnaître des images mais elles doivent être très similaires les unes des autres.

La particularité d'un dépôt d'ordure, c'est qu'il est de taille, de couleur et de composition variable.

On va donc se diriger vers des technologies de deep learning qui sont plus performantes.

Les objectifs du projet

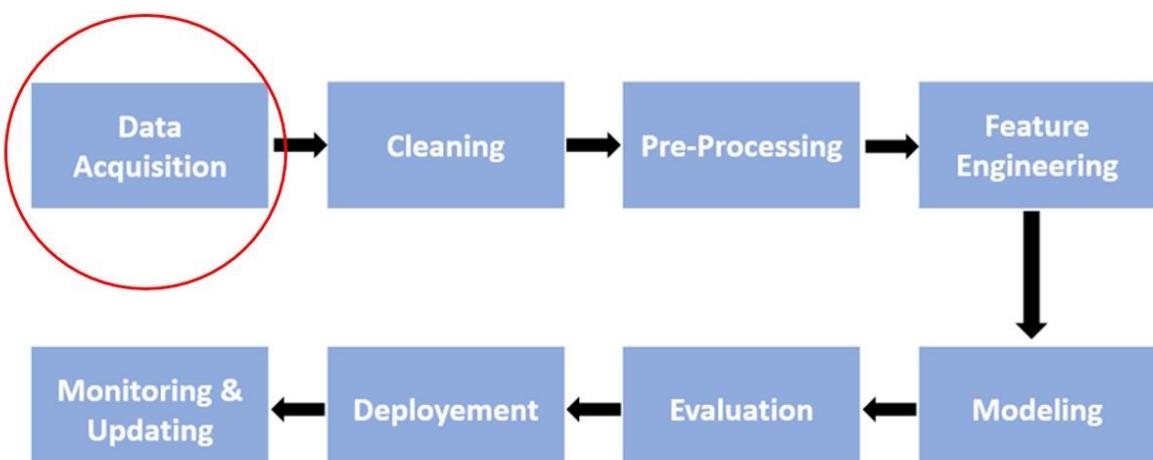
Le pipeline de la donnée du projet est le suivant :



Evidemment, il est idéal et il n'est pas certain qu'on puisse faire du feature engineering sur les images si on n'a pas de variables.

On va procéder par ordre chronologique et commencer avec l'acquisition de la base de données.

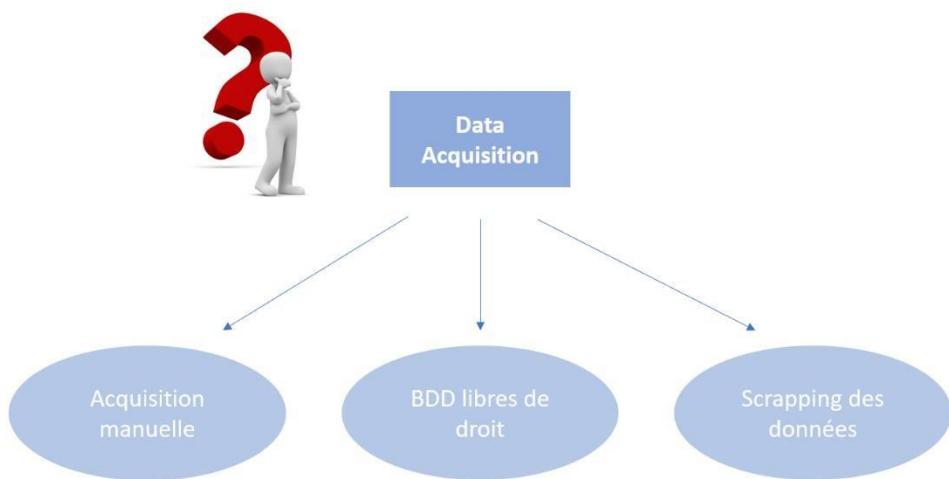
Data Acquisition



Une première question se pose : comment récupérer la base de données ?

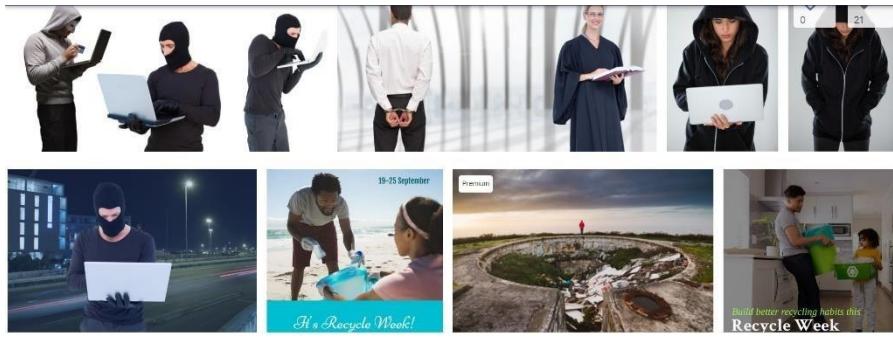
Dans notre cas, on n'a pas de base de données et l'entreprise nous demande de la constituer nous-même. C'est une étape délicate car la base de données doit être de qualité : si une base de données n'est pas représentative, l'entraînement du modèle se fera sur des données biaisées ou fausse, donc il sera inutilisable. On a intérêt à nourrir une base de données la plus propre possible.

Trois solutions s'offrent à nous pour le remplissage :



Acquisition manuelle : consiste à prendre soit même des photos de dépôts d'ordures illégaux. Ce serait la solution qui offrirait les images de la meilleure qualité, mais malheureusement c'est irréalisable car on a besoins de centaines d'images pour notre projet.

Les BDD libres de droit : c'est une solution avantageuse vers laquelle on pensait se tourner au début du projet. Après quelques recherches, on a décidé de ne pas les utiliser car elles n'étaient pas de bonne qualité. Quelques exemples :

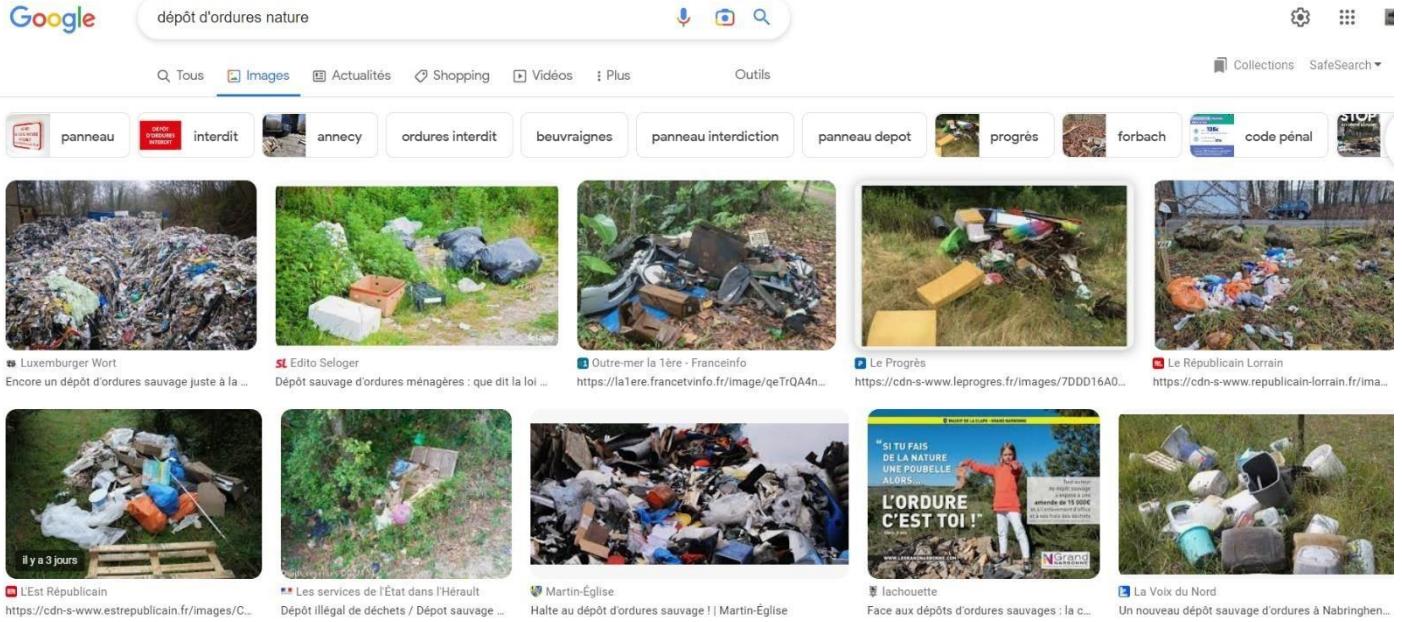


Ci-dessus, les résultats de la recherche avec **Pikwizard**, une base de données connues libre de droits, en utilisant le mot clé « illégal dump » c'est-à-dire « dépôt d'ordures illégal » en anglais. Les images ne représentent clairement pas ce que nous cherchons.



La même chose avec **VisualHunt**, toujours avec le même mot clé. Les images se rapprochent du résultat attendu mais sont toujours trop approximatives pour constituer une BDD de qualité.

Une recherche google, elle, donne des résultats bien meilleurs :



C'est bien, mais c'est très long de récupérer des centaines d'images à la main. C'est ce qui nous a poussé à nous intéresser aux techniques de scraping des données.

Le scraping, littéralement « grattage » ou « raclement » consiste à utiliser un algorithme qui automatise la recherche et la récupération de données. Dans notre cas, il faut mettre au point un algorithme qui récupère nos images sur google images et qui les télécharge dans un dossier.

Scraping des données : comment procéder ?

Pour scraper des données, le framework le plus ancien est selenium. Comme il est sorti en 2002, c'est aussi le plus documenté.

D'autres librairies existent, comme Scrappy, Jaunt, ou Mechanical Soup.

Par économie de temps, c'est Selenium qu'on va utiliser.

Il permet d'interagir avec des navigateurs web comme google chrome ou Mozilla Firefox en utilisant les drivers web, comme le ferait un utilisateur.

C'est un outil de test qui est exécutable dans de nombreux langages (java, php), et même en python.



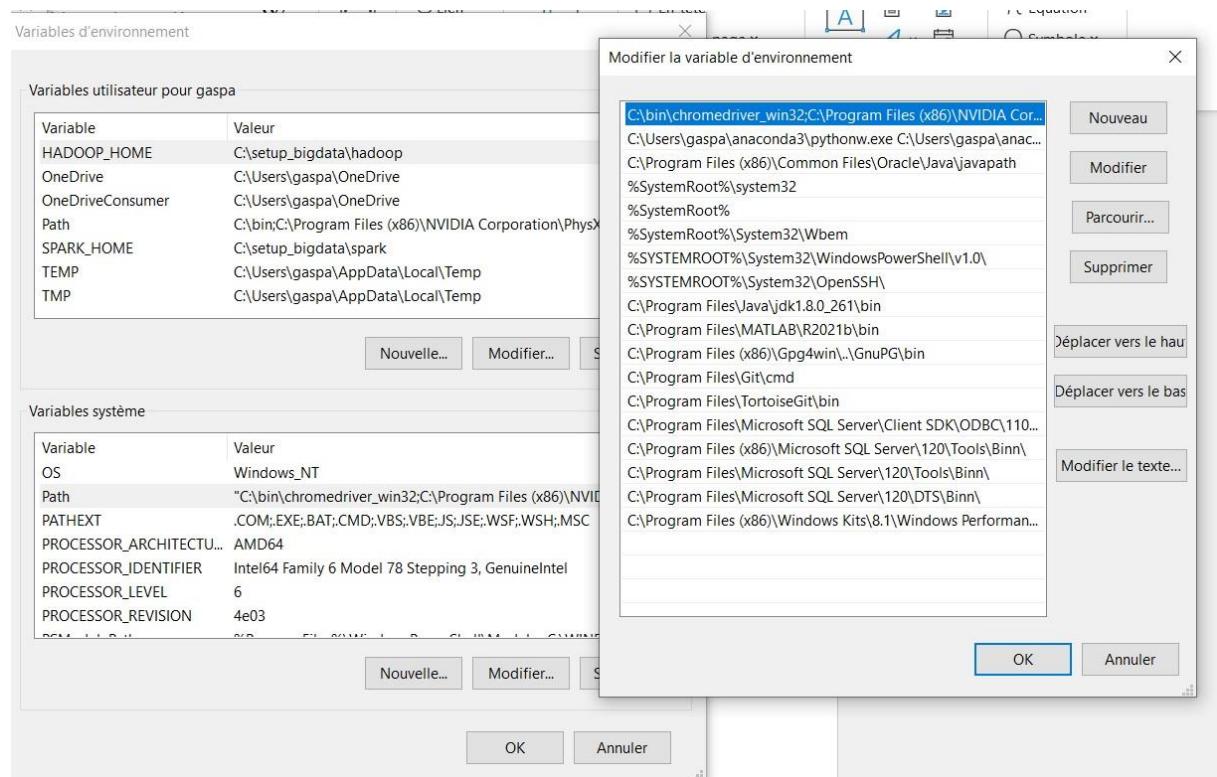
Tout d'abord, il faut télécharger les drivers. Google chrome donne des meilleures images que mozilla, on va donc utiliser les chromedriver. Si on passait par mozilla, on aurait utilisé les drivers correspondant à ce navigateur, qui sont les geckodrivers.

Les chromedrivers peuvent être trouvés sur internet sous forme de zip. Lors de l'extraction, il faut désactiver le pare feu et le proxy car c'est un fichier exe de source inconnue. On peut aussi l'ouvrir en mode administrateur pour contourner ce problème.

Pour faire fonctionner le chromedriver, il faut le placer dans un dossier bin dans lequel on créé notre fichier scrapping_google.ipynb.

On a choisi le format ipynb car il permet de tester le code en permanence et c'est très pratique quand on veut pouvoir observer les résultats presque en temps réel.

Pour faire fonctionner les drivers, il faut également créer une variable d'environnement système qui pointe sur le dossier contenant les chromedrivers :



On ouvre l'administrateur de variables d'environnement et on ajoute une nouvelle variable à PATH. On fait pointer cette nouvelle variable vers le chemin du dossier contenant les chromedrivers.

Une fois cette étape réalisée, on ouvre notre fichier ipynb et on importe les librairies qui vont nous être utiles :

```
In [5]: # doc officielle de selenium: https://selenium-python.readthedocs.io/index.html
# lien pour installer le webdriver à coup sûr: https://chromedriver.chromium.org/getting-started
# selon la version de selenium, les webdriver ne sont pas compatibles avec google collab.
# il faut enregister ce fichier dans le même dossier que chromedriver.exe

# !pip install selenium
# !pip install web_driver_manager
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

from selenium.webdriver.common.keys import Keys
import os
import json
import urllib
import sys
import time
```

Selon la version de selenium utilisée, les webdrivers peuvent ne pas être compatibles avec google collab. Après quelques recherches sur StackOverflow, on a aussi remarqué que selon la version de selenium, le webdriver peut ne pas fonctionner et il faut l'importer de la librairie. De nouveau, plusieurs versions existent et se chevauchent, créant des erreurs d'exécution. Les versions à installer peuvent être webdriver-manager, webdriver_manager, ou encore web-driver-manager.

Ensute, on configure les variables globales à instancier :

```
In [11]: #on importe les driver et on se connecte à google
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
driver.get("https://www.google.com")
options = webdriver.ChromeOptions()

##pour régler le problème d'interactibilité ??
options.add_argument("start-maximized")
options.add_argument("disable-infobars")
options.add_argument("--disable-extensions")

# on implémente le path vers geckodriver vers la variable d'environnement OS
os.environ["PATH"] += os.pathsep + os.getcwd()

# Configuration
download_path = "dataset/"
# Images
words_to_search = ['Dépôt d\'ordures nature','nature champs']
nb_to_download = [30,30]
#number = nb_to_download[0]
first_image_position = [10,10]
#first_position = first_image_position[0]
```

Selenium automatise les actions qu'on lui demande d'effectuer. Il faut donc lui donner toutes les informations nécessaires pour qu'il se connecte sans encombre à la page qu'on souhaite.

On rencontre bien sûr de nombreuses erreurs, le code d'un algorithme comme celui-ci nécessite beaucoup de recherches et de tâtonnement.

On précise les mots à chercher, la position de la première image à télécharger, le nombre d'images à télécharger (par liste, car chaque mot clé correspond à une liste d'images).

On configure le main qui renvoie une erreur si l'exception est franchie.

```
def main():
    if len(words_to_search) != len(nb_to_download) or len(nb_to_download) != len(first_image_position) :
        raise ValueError('You may have forgotten to configure one of the lists (length is different)')
    i = 0
    # pour chaque mot dans la liste, on télécharge le nombre d'images
    while i < len(words_to_search):
        print("Words "+str(i)+" : "+str(nb_to_download[i])+"x\""+words_to_search[i]+"\"")
        if nb_to_download[i] > 0:
            search_and_save(words_to_search[i],nb_to_download[i], first_image_position[i])
        i+=1
```

On définit la fonction search_and_save qui va s'occuper de trouver et de télécharger les images dont on a besoin.

```
def search_and_save(text, number, first_position):
    # Number_of_scrolls * 400 images will be opened in the browser
    number_of_scrolls = int((number + first_position)/ 400 + 1)
    print("Search : "+text+" ; number : "+str(number)+" ; first_position : "+str(first_position)+" ; scrolls : "+str(number_of_scrolls))

    # on crée les répertoires pour enregistrer les images
    if not os.path.exists(download_path + text.replace(" ", "_")):
        os.makedirs(download_path + text.replace(" ", "_"))

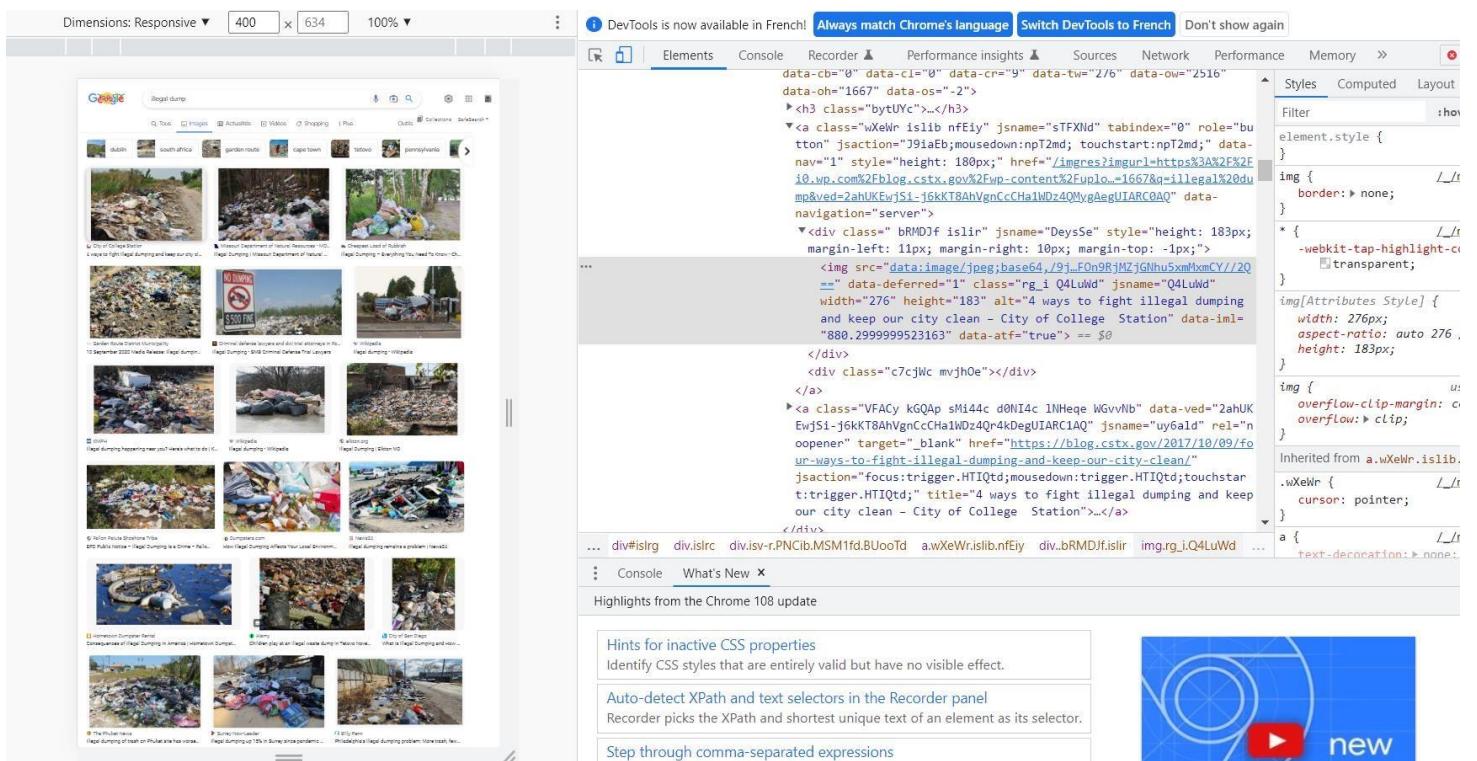
    # on se connecte à google image
    url = "https://www.google.co.in/search?q="+text+"&source=lnms&tbo=isch"
    driver = webdriver.Chrome()
    driver.get(url)
    # ne pas oublier de dire à notre script de cocher la case "tout accepter" concernant les cookies
    driver.find_element(By.XPATH, "//*[@id='yDmH0d']/c-wiz/div/div/div[2]/div[1]/div[3]/div[1]/div[1]/form[2]/div/div/button/span").click()
    headers = {}
    headers['User-Agent'] = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36"
    extensions = {"jpg", "jpeg", "png", "gif"}

    img_count = 0
    downloaded_img_count = 0
    img_skip = 0

    # on prépare la page google
    for _ in range(number_of_scrolls):
        for __ in range(10):
            # on multiplie par le nombre de scrolls
            driver.execute_script("window.scrollBy(0, 1000000)")
            time.sleep(0.2)
        # pour charger 400 images
        time.sleep(2.5)
        try:
            driver.find_element(By.XPATH, "/html/body/div[2]/c-wiz/div[3]/div[1]/div/div/div/div[1]/div[2]/div[2]/input").click()
            time.sleep(2.5)
        except Exception as e:
            print("Less images found:"+ str(e))
            break
```

Il faut préciser toutes les actions qu'un utilisateur ferait pour télécharger une image : le nombre de scrolls de la souris (suffisamment pour descendre tout en bas de la page google image), il faut également lui apprendre à gérer les obstacles, comme cliquer sur le bouton « accepter les closes de confidentialité google », sur le bouton « charger plus d'images », qui sont des balises HTML.

Pour les récupérer, on se rend sur google image et on récupère le XPath du container des vignettes des images en inspectant le code source de la page.



On peut utiliser d'autres balises, comme le Jsname, le totalPath ; l'important, c'est d'avoir une clé unique qui identifie les images par un indice sur lequel on peut boucler.



Philadelphia Inquirer
Miles of trash: Litter detectives stymied by ille...



Northern Inland Regional Waste
Illegal Dumping - Northern Inland Regional Wa...



OurAuckland - Auckland Council
Illegal dumper faces full force of law - OurAuc...

Afficher plus de résultats

On réitère l'opération pour que notre algorithme trouve et sélectionne les balises cliquables.

```
nibp = 400
images=[]
for i in range(number_of_scrolls*nibp):
    chemin = ('/html/body/div[2]/c-wiz/div[3]/div[1]/div/div/div/div[1]/div[1]/span/div[1]/div[1]/div['+str(i+1)+']/a[1]/div[1]/img')
    img = driver.find_element(By.XPATH, chemin)
    print("img:", img)
    images.append(img)
print("Total images:"+ str(len(images)) + "\n")
```

Ci-dessus, on boucle sur le chemin du total XPath pour récupérer chaque image présente sur google image.

Un autre problème fait barrage au scraping : les grandes sociétés comme Meta, Alphabet, qui possèdent Facebook, google, etc rendent la tâche difficile au scraping des données. Comme ils veulent vendre leurs propres algorithmes et leurs propres bases de données, ils changent régulièrement le code source de leurs pages HTML sans aucune autre raison que de gêner les algorithmes de scraping qui ne reconnaissent plus les balises.

Il faut dans un premier temps utiliser une fonction de sleep : elle marque des pauses entre les étapes et de cette manière google ne repère pas que c'est un ordinateur qui fait la recherche et qui franchit des vitesses excessives.

Google change aussi régulièrement le XPath : entre plusieurs essais, en plus des erreurs de code qu'on doit corriger, on doit de nouveau récupérer le XPath en allant chercher la balise des vignettes images.

<https://www.webrankinfo.com> › ... › Google Search

Google veut lutter contre le scraping et appelle à la dénonciation

29 août 2011 — Google vient de mettre en ligne un formulaire dédié à la dénonciation des scrapers.

Recherches associées

google ban scraping google scraping legal
scrape google search results python serpmaster
web scraping without getting blocked google search scraper

<https://serpmaster.com> › blog › scra... · Traduire cette page

Scrape Google Without Getting Blocked – 8 Useful Tips

25 mars 2022 — Learn about web scraping best practices that will help you to avoid facing blocks while scraping Google.

<https://news.ycombinator.com> › item · Traduire cette page

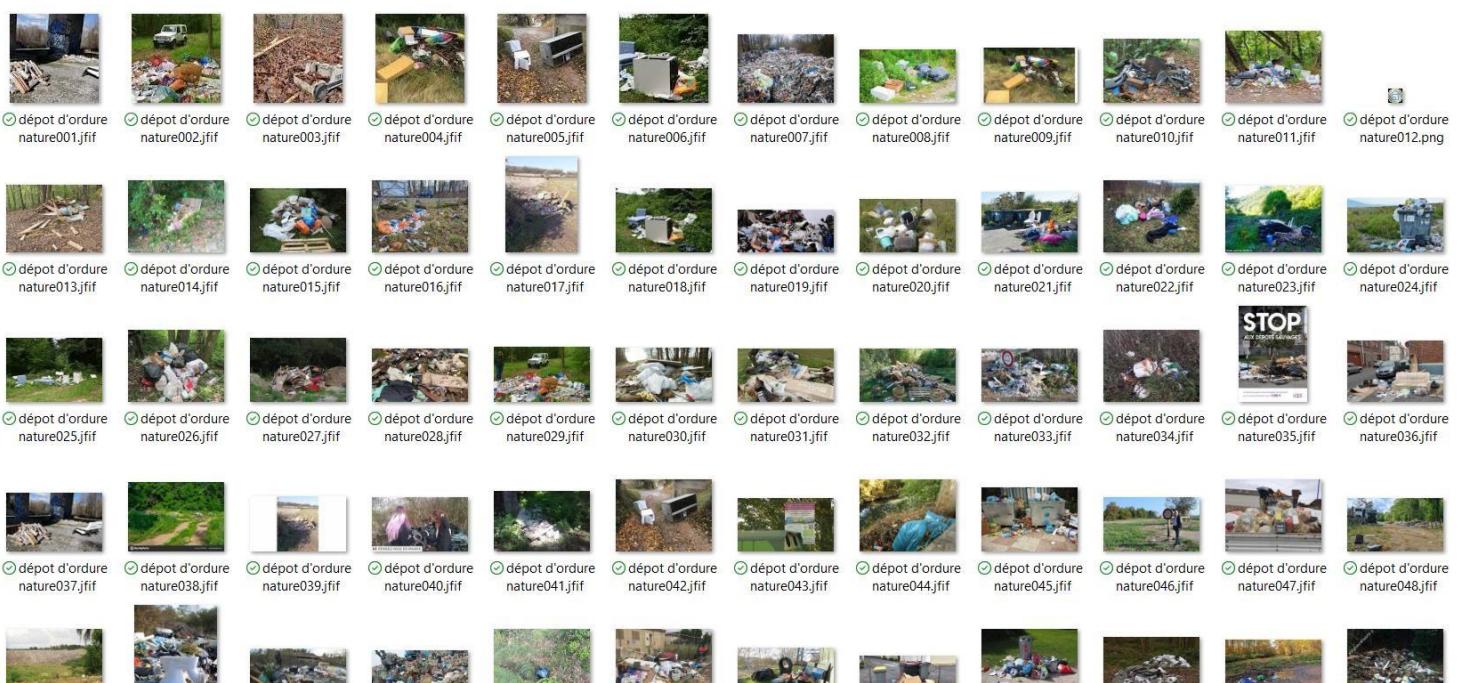
There is an irony in google preventing web scraping given that ...

Why is there irony in that? Anyone can go build a crawler and scrape the web the way Google scrapes it so they can compete with Google.

Autres questions posées :

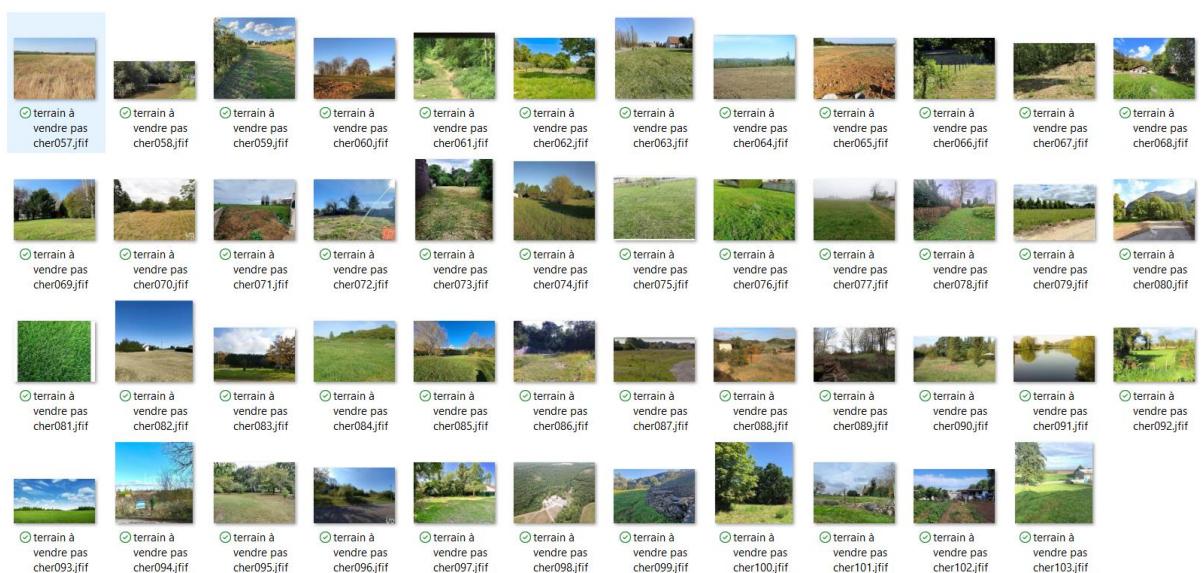
Pourquoi Google utilise T-IL le scraping ?

On récupère les images dans un dossier créé par notre algorithme où elles sont identifiées par un numéro



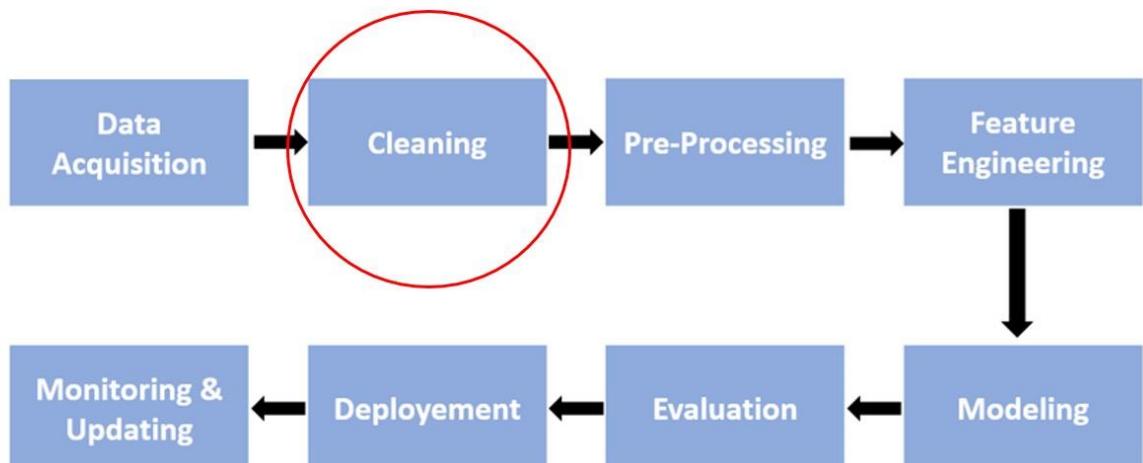


Une fois notre classe `depot_nature` obtenue, il nous faut constituer la deuxième classe. Cette classe doit être représentative de photos amateurs prises dans la France rurale, sur des pistes, en forêt ou bien sur des départementales. Les mots « nature » ou « France rurale » ne sont pas très pertinents, donc il faut trouver des termes plus techniques. Avec les mots « friche », « terrain pas cher » et « terrain paca », on obtient des résultats pertinents : on croirait voir les images de dépôt d'ordure sans les ordures, c'est exactement ce qu'il nous faut. On réalise de nouveau un scrapping pour ces trois termes et on obtient les images suivantes :





Cleaning



Certaines images polluent notre BDD et n'ont rien à y faire :



Nombre d'entre elles sont des caricatures, des arrêtés municipaux, des affiches de prévention contre la pollution. Malheureusement, la seule manière de les effacer, c'est de les supprimer manuellement.

C'est une étape longue et fastidieuse mais elle est nécessaire : empiriquement, on sait que si la base de données est nourrie avec des données de mauvaise qualité, l'entraînement qui se fera sur ces données rendra des prédictions de mauvaise qualité.

Data augmentation

Maintenant qu'on a une base de données exploitable, on est face à un problème : elle ne contient que 284 images pour la classe « dépôt d'ordure nature ».

Pour l'entraînement, il serait bon d'avoir plus de données. Les stratégies d'augmentation de donnée permettent de transformer, brouiller, et modifier des images pour augmenter la taille de la BDD en conservant une certaine rigueur dans la qualité des images.

Il s'agit d'appliquer un traitement sur chaque image pour en faire des variantes viables.

Pour ce faire, nous nous proposons d'utiliser le package transforms de la librairie torch.

Le notebook Jupyter est assez court :

```

Entrée [8]: import torchvision
import torch
import torchvision.transforms as transforms
import os
import matplotlib.pyplot as plt
import numpy as np

Entrée [15]: train_dataset_path = r"C:\bin\database_train"
test_dataset_path = r"C:\bin\database_test"

Entrée [16]: mean = [0.4363, 0.4328, 0.3291]
std = [0.2129, 0.2075, 0.2038]

train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(torch.Tensor(mean), torch.Tensor(std)),
])

test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(torch.Tensor(mean), torch.Tensor(std)),
])

Entrée [17]: train_dataset = torchvision.datasets.ImageFolder(root = train_dataset_path, transform = train_transforms)
test_dataset = torchvision.datasets.ImageFolder(root = test_dataset_path, transform = test_transforms)

```

Il faut récupérer la moyenne et l'écart type de nos images pour les normaliser.

```

Entrée [22]: def show_transformed_images(dataset):
    loader = torch.utils.data.DataLoader(dataset, shuffle=True)
    batch = next(iter(loader))
    images, labels = batch

    grid = torchvision.utils.make_grid(images, nrow = 3)
    plt.figure(figsize=(11, 11))
    plt.imshow(np.transpose(grid, (1, 2, 0)))
    print("labels:", labels)

Entrée [23]: show_transformed_images(train_dataset)

```

C:\Users\gaspa\anaconda3\lib\site-packages\PIL\Image.py:975: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
 warnings.warn(
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
labels: tensor([0])

Toutefois, nous n'avons pas obtenu de bons résultats. C'est encore en développement.

Finalement, nous nous sommes heurtés à un échec en utilisant la librairie torch.

Du coup, on s'est tourné vers la librairie ImgAug.

Imgaug est une bibliothèque qui permet de mettre en pratique les techniques d'augmentation d'images dans les projets relatifs à l'apprentissage automatique. Elle prend en charge un large éventail de techniques d'augmentation.

Ces techniques consistent là aussi en des opérations de rotation, de bruitage, de contraste des images...

Imgaug permet de les combiner facilement et de les exécuter dans un ordre aléatoire et peut non seulement augmenter les images, mais aussi les points clés/marques de terrain, les boîtes de délimitation, les cartes thermiques et les cartes de segmentation.

```
> pip install imgaug
```

```
import imageio
import imgaug as ia
from imgaug import augmenters as iaa
import os
```

```
folder_path = r"C:\Users\gaspa\Downloads\frique" #chemin du dossier des images source
folder_path_ = r"C:\Users\gaspa\Downloads\frique_transformed" #chemin du dossier des images transformées

# Definition des augmentations
seq = iaa.Sequential([
    iaa.Flipud(), # flip vertical
    iaa.Affine(rotate=(-45, 45)), # rotation de 45 degrés
    iaa.AdditiveGaussianNoise(scale=(0, 0.1*255)), # ajout de bruit
    iaa.Multiply((0.5, 1.5)) # modification de la brillance
    #zoom iaa.Affine(scale{10, 20})
    #flip vertical 50% proba: iaa.Fliplr(0.5)
    #diminution grossière: iaa.CoarseDropout(p=0.5, size_px=int, size_percent=None, per_channel=False)
    #iaa.ElasticTransformation(alpha=int, sigma=int, order=int) transformation élastique
    #iaa.GammaContrast((0.5, 2.0)) normalisation de contraste
])

# on itère pour toutes les images du dossier
for filename in os.listdir(folder_path):
    # on opère sur les fichiers images
    if not os.path.isdir(filename):
        # on charge les images
        image = imageio.imread(os.path.join(folder_path, filename))
        # on exécute les augmentations
        augmented_image = seq.augment_image(image)
        # on sauvegarde les images augmentées
        imageio.imwrite(os.path.join(folder_path_, "augmented_" + filename), augmented_image)
```

Ci-dessus, le code commenté explique assez bien le fonctionnement de l'algorithme. On sélectionne un répertoire source et un répertoire cible. Ensuite, on définit une fonction seq qui va appliquer différents opérateurs à d'augmentation à nos images, parmi ceux disponibles dans la librairie. Ensuite, on boucle sur tous les fichiers du dossier source pour appliquer les opérateurs aux images.

On a choisi les opérateurs qui permettaient de modifier l'image tout en restant fidèle au visage de la base de données. Voici des exemples de transformation à partir d'une image de friche.



Image source



Modifications de la brillance



Ajout d'un bruit gaussien



Contraste gamma



Rotation aléatoire



Flip aléatoire

Etc...

Une fois la sélection des opérateurs effectués, on boucle sur toute la base de données. On passe d'une BDD de 280 images par classe à plus de 500 images par classe.

Pre Processing

Il y a une réflexion qui est en cours sur la suite du projet : avec la segmentation, on peut obtenir des résultats de 70% à 80% avec un dataset de 284 images actuelles, voire 500 avec l'augmentation de données.

Si on reste sur de la segmentation, il est important de bien choisir les classes : on peut dresser une liste du type d'environnement à proximité du tas d'ordure pour nous aider à faire le choix des classes.

On peut faire un labelling très précis avec beaucoup de classes différentes puis faire des tests d'entraînement avec différentes combinaisons d'union.

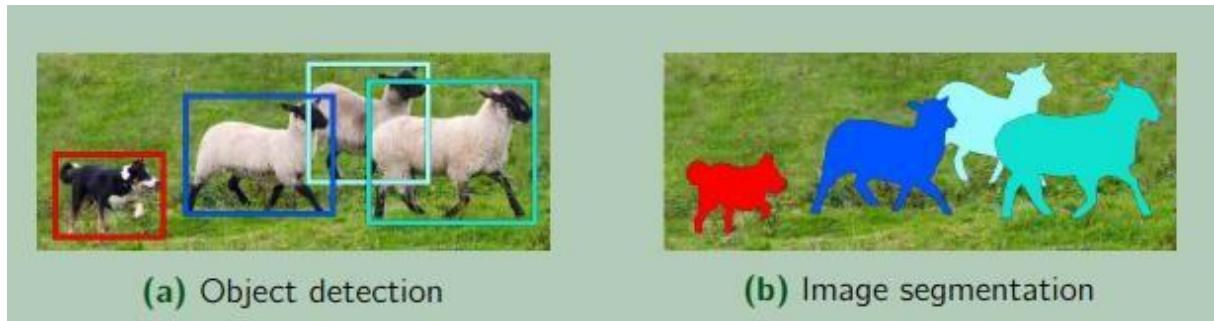
Pour être plus clair, on peut créer des classes primaires : arbres, sol herbeux, route, chemin, ciel, nuage, tas d'ordure.

Ensuite, on fait une union de classes : végétation = arbre + sol herbeux, route = route + chemin, ciel = ciel + nuage.



Ainsi, on peut tester différentes combinaisons de classe sans avoir à tout tester, mais, en contrepartie, le labelling va prendre plus de temps.

On peut aussi faire de l'identification d'objets fréquemment présents dans des tas d'ordure : pneus, carcasses de voitures, bouteille en plastique, cannette, poubelles...



L'autre possibilité, c'est la détection d'objet avec un réseau de deep learning.

Pour le traitement qui suit, on va beaucoup utiliser la bibliothèque OS en python. OS est une bibliothèque qui permet d'interagir avec le système d'exploitation du PC (mac, windows, linux) et effectuer des opérations courantes tel que : créer des dossiers, gérer des fichiers, obtenir des informations sur les fichiers, déplacer, renommer, supprimer, etc, de manière massive et automatique, en quelques lignes de code.



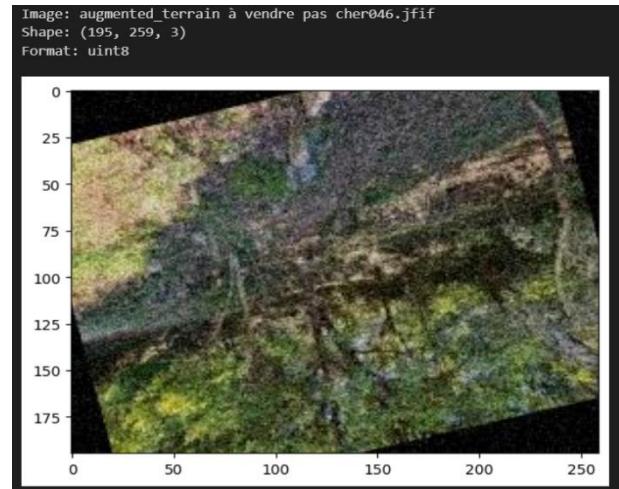
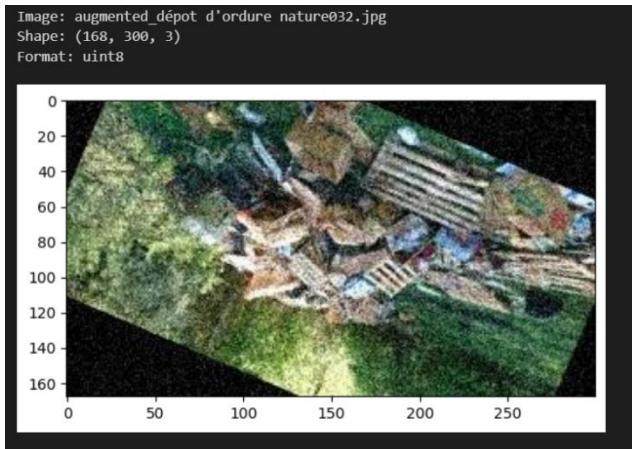
On a un autre problème concernant le format, et la taille des images. En effet, en chinant dans la base de données, on s'aperçoit que toutes les images n'ont pas la même extension, et n'ont pas toutes le même format. On s'en rend facilement compte en codant un algorithme python qui renvoie aléatoirement 10 images de notre base de données :

```
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Récupérer la liste des noms de fichiers d'images
image_files = os.listdir(train_dir)

# Sélectionner 10 fichiers d'images de manière aléatoire
selected_images = random.sample(image_files, 10)

# Afficher les images sélectionnées avec leur shape et leur format
for image_file in selected_images:
    image_path = os.path.join(train_dir, image_file)
    image = mpimg.imread(image_path)
    print('Image:', image_file)
    print('Shape:', image.shape)
    print('Format:', image.dtype)
    plt.imshow(image)
    plt.show()
```

✓ 5.6s



Comme on peut le voir ci-dessus, certaines images ont une shape (length, width, dim) de (168, 300, 3), d'autres ont (183, 276, 3), et d'autres (174, 290, 3), et d'autres encore. De plus, la plupart des images téléchargées sont au format jfif, ce qui va nous poser des problèmes au moment de l'entraînement. Il va donc falloir harmoniser la shape et le format des images dans notre base de données.

Il faut connaître tous les formats pour que l'algorithme de notre loop puisse boucler sur tous les fichiers. Pour ceci, on écrit un petit algorithme qui permet de récupérer tous les formats de fichiers présents dans les dossiers de la base de données.

```

> ^      import os

path = r"C:\Users\gaspa\Downloads\database_2\test" # Chemin vers le dossier à inspecter
extensions = set() # Initialisation d'un ensemble pour stocker les extensions de fichier uniques

for file_name in os.listdir(path): # Parcours de tous les fichiers dans le dossier
    extension = os.path.splitext(file_name)[1] # Récupération de l'extension du fichier
    extensions.add(extension) # Ajout de l'extension à l'ensemble

print("Extensions de fichiers présentes dans le dossier {}: {}".format(path, extensions))
[32]  ✓  0.0s
... Extensions de fichiers présentes dans le dossier C:\Users\gaspa\Downloads\database_2\test: {'.jpg', '.png', '.jfif'}
  
```

Il y a donc trois formats d'image dans notre base de données : jpg, png, jfif.

Or, en lançant un entraînement de manière prématurée, on se rend compte que le format jfif n'est pas supporté par le CNN. Il faut donc convertir toutes les images dans un format supporté : jpeg, png...

On a également un autre problème. Certaines images portent le nom suivant : « dépôt d'ordure nature1 », « dépôt d'ordure nature2 », etc. En tentant de convertir toutes les images en png, on s'est rendu compte que les accents et les apostrophes dans les noms de fichier posent problème.

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
import os
import cv2

# Définir les dossiers source et destination
src_folder = r"C:\Users\gaspa\Downloads\database_2\train"
dst_folder = r"C:\Users\gaspa\Downloads\database_3\train"

# Taille souhaitée pour les images
target_size = (100, 100)

# Itérer sur toutes les images dans le dossier source
for filename in os.listdir(src_folder):
    # Vérifier que le fichier est une image
    if filename.endswith('.jpg') or filename.endswith('.jfif') or filename.endswith('.png'):
        # Charger l'image
        img = cv2.imread(os.path.join(src_folder, filename))
        img_path = os.path.join(src_folder, filename)
        # Redimensionner l'image
        if img is not None:
            img_resized = cv2.resize(img, target_size)
        # Enregistrer l'image redimensionnée dans le dossier destination
        cv2.imwrite(os.path.join(dst_folder, filename), img_resized)
    else:
        print(f"Erreur de lecture de l'image: {img_path}")

[40] 0.2s
```

... Output exceeds the size_limit. Open the full output data in a text editor

Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature001.jfif
Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature001.jpg
Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature002.jfif
Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature002.jpg
Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature003.jfif
Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature003.jpg
Erreurs de lecture de l'image: C:\Users\gaspa\Downloads\database_2\train\augmented_dépot d'ordure nature004.jfif

Python

Jupyter Server: Local Cell 7 of 9

Ci-dessus, on a créé un algorithme pour resizer les images à la taille (100, 100, 3), en itérant sur un dossier source pour envoyer les nouvelles images vers un dossier cible. Or, en sortie, on constate que l'exception a été levée : il y a une erreur de lecture de l'image. En observant la sortie, on se rend compte qu'en fait, le « é » de « dépôt d'ordure nature » est la source du problème.

Il faut donc réécrire les noms des fichiers de sorte qu'ils ne contiennent ni accent, ni apostrophes, ni aucun caractère spécial qui pourrait gêner les boucles des algorithmes.

Pour cela, on a écrit le script suivant dans notre jupyter notebook:

cnn.ipynb ● data_augmentation.ipynb ● gaspard_saint_upery_tp2_deep_learning (1) (2).ipynb

C:\> Users > gaspa > OneDrive > Bureau > cours 4ème année > projet > cnn.ipynb > import os

+ Code + Markdown | Run All Clear All Outputs Restart Variables Outline ... > lr Aa ab .* No results ↑ ↓ ×

error: OpenCV(4.7.0) D:\a\opencv-python\opencv-python\opencv\modules\imgcodecs\src\loadsave.cpp:692: error: (-2:unspecified error) could not find a writer for the specified extension in function 'cv::imwrite_'

```

import os

# Chemin vers le dossier contenant les éléments à renommer
folder_path = r"C:\Users\gasp\Downloads\test_\source"

# Nom commun à utiliser pour tous les éléments renommés
new_name = "train"

# Compteur pour le numéro d'ordre
count = 1

# Boucle pour renommer tous les éléments
for old_name in os.listdir(folder_path):
    # Créer le nouveau nom avec le numéro d'ordre
    new_name_with_count = new_name + str(count) + ".png"

    # Chemin complet vers l'élément à renommer
    old_path = os.path.join(folder_path, old_name)

    # Chemin complet vers l'élément renommé
    new_path = os.path.join(folder_path, new_name_with_count)

    # Renommer l'élément
    os.rename(old_path, new_path)

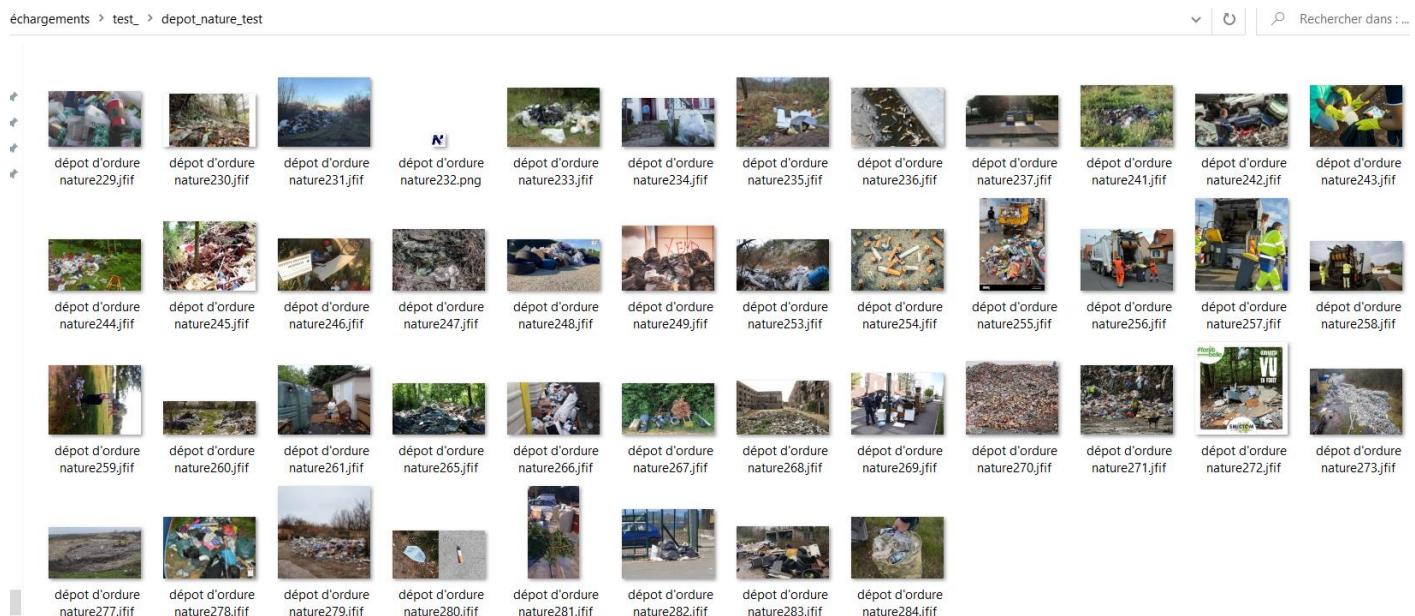
    # Incrémenter le compteur
    count += 1

```

[47] ✓ 0.1s Python

Comment fonctionne ce script ? C'est très simple ! On spécifie le folder source et on initialise un compteur qu'on va instancier à chaque itération. Avec la bibliothèse os, on peut renommer l'élément automatiquement avec un nom « new_name » défini selon notre dossier.

Prenons un exemple avec le dossier depot_nature_test suivant :

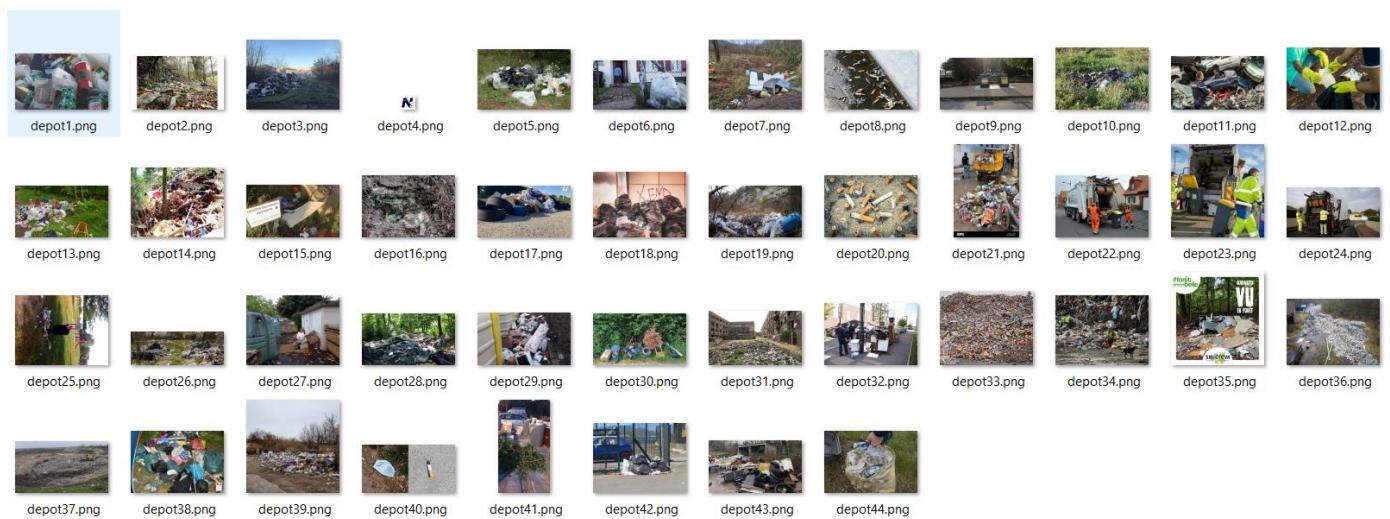


Comme on peut le remarquer, il contient des noms comportant des accents, des apostrophes, etc. On spécifie le nom de ce dossier dans le script :

```
# Chemin vers le dossier contenant les éléments à renommer
folder_path = r"C:\Users\gaspa\Downloads\test_\depot_nature_test"

# Nom commun à utiliser pour tous les éléments renommés
new_name = "depot"
```

On se retrouve alors avec le dossier suivant :



De cette manière, on a réglé tous nos problèmes concernant le nom des fichiers. Ce qui veut dire qu'on peut désormais utiliser notre algorithme de redimensionnement sans problème !

Bien qu'on puisse passer à la suite, on remarque quand même quelque chose d'étrange quand on veut afficher certaines de ces images :

```
...
Traceback (most recent call last):
File c:\Users\gaspa\AppData\Local\Programs\Python\Python310\lib\site-packages\IPython\core\interactiveshell.py:3398 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
Input In [54] in <cell line: 14>
    image = mpimg.imread(image_path)
File c:\Users\gaspa\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\image.py:1541 in imread
    with img_open(filename) as image:
File c:\Users\gaspa\AppData\Local\Programs\Python\Python310\lib\site-packages\PIL\ImageFile.py:117 in __init__
    self._open()
File c:\Users\gaspa\AppData\Local\Programs\Python\Python310\lib\site-packages\PIL\PngImagePlugin.py:732 in _open
    raise SyntaxError(msg)
File <string>
SyntaxError: not a PNG file
```

En fait, certaines images PNG ne sont pas reconnues par notre algorithme. Lesquelles ?

```
# Afficher les images sélectionnées avec leur shape et leur format
for image_file in selected_images:
    image_path = os.path.join(v_path, image_file)
    try:
        image = mpimg.imread(image_path)
        print('Image:', image_file)
        print('Shape:', image.shape)
        print('Format:', image.dtype)
        plt.imshow(image)
        plt.show()
    except:
        print(f"{image_file} n'est pas un fichier PNG valide.")

[69]    ✓ 0.0s
...
depot1.png n'est pas un fichier PNG valide.
depot35.png n'est pas un fichier PNG valide.
depot6.png n'est pas un fichier PNG valide.
depot17.png n'est pas un fichier PNG valide.
depot25.png n'est pas un fichier PNG valide.
depot34.png n'est pas un fichier PNG valide.
depot19.png n'est pas un fichier PNG valide.
depot7.png n'est pas un fichier PNG valide.
depot8.png n'est pas un fichier PNG valide.
depot43.png n'est pas un fichier PNG valide.
```

En ajoutant l'exception ci-dessus, on se rend compte qu'environ $\frac{1}{4}$ de nos images ne sont pas des PNG valides, alors qu'elles sont bien enregistrées au format PNG. Ce qui signifie que ces fichiers sont **corrompus**. Ce problème peut venir du téléchargement des images au moment du scrapping, ou bien d'une taille trop petite. Les dossiers corrompus pourraient poser un problème lors de l'entraînement, toutefois, comme on va le voir, le redimensionnement à lui seul permet d'effacer ce problème de fichiers corrompus. On n'a pas creusé le sujet plus loin, mais c'est un point intéressant à soulever, qui mérite qu'on s'y intéresse.

```

: > Users > gaspa > OneDrive > Bureau > cours 4ème année > projet > cnn.ipynb >      ... > lr

Code + Markdown | Run All Clear All Outputs Restart Variables Outline ...
> lr

> ###CODE POUR AFFICHER LA SHAPE DE 10 IMAGES ALEATOIRES DANS LE DOSSIER

v_path = r"C:\Users\gaspa\Downloads\test_\cible"
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Récupérer la liste des noms de fichiers d'images
image_files = os.listdir(v_path)

# Sélectionner 10 fichiers d'images de manière aléatoire
selected_images = random.sample(image_files, 10)

# Afficher les images sélectionnées avec leur shape et leur format
for image_file in selected_images:
    image_path = os.path.join(v_path, image_file)
    image = mpimg.imread(image_path)
    print('Image:', image_file)
    print('Shape:', image.shape)
    print('Format:', image.dtype)
    plt.imshow(image)
    plt.show()

[60] ✓ 2.4s
... Image: depot19.png
Shape: (100, 100, 3)
Format: float32

```

On obtient des images de shape (100, 100, 3)

Voici un code qui nous permet de vérifier la shape de chacune des images du dossier :

```

from PIL import Image
import os

folder_path = r"C:\Users\gaspa\Downloads\test_\cible"

for filename in os.listdir(folder_path):
    if filename.endswith('.png'):
        image_path = os.path.join(folder_path, filename)
        with Image.open(image_path) as img:
            print(f"La forme de l'image {filename} est {img.size}")

[63] ✓ 0.0s
... Output exceeds the size limit. Open the full output data in a text editor
La forme de l'image depot1.png est (100, 100)
La forme de l'image depot10.png est (100, 100)
La forme de l'image depot11.png est (100, 100)
La forme de l'image depot12.png est (100, 100)
La forme de l'image depot13.png est (100, 100)
La forme de l'image depot14.png est (100, 100)
La forme de l'image depot15.png est (100, 100)
La forme de l'image depot16.png est (100, 100)
La forme de l'image depot17.png est (100, 100)
La forme de l'image depot18.png est (100, 100)
La forme de l'image depot19.png est (100, 100)
La forme de l'image depot2.png est (100, 100)
La forme de l'image depot20.png est (100, 100)
La forme de l'image depot21.png est (100, 100)
La forme de l'image depot22.png est (100, 100)
La forme de l'image depot23.png est (100, 100)
La forme de l'image depot24.png est (100, 100)
La forme de l'image depot25.png est (100, 100)
La forme de l'image depot26.png est (100, 100)
La forme de l'image depot27.png est (100, 100)
La forme de l'image depot28.png est (100, 100)
La forme de l'image depot29.png est (100, 100)

```

Classification

Maintenant qu'on a nos images et que la base de données est constituée, on peut passer à l'entraînement.

Notre base de données contient **523 images de dépôts et 515 images de friche, soit 1038 images au total.**

Comme on peut le constater, la base de données est équilibrée par rapport à chaque classe, on n'aura pas de problème de spécialisation à l'entraînement à ce niveau-là.

Pour rappel, voici comment se présente l'arborescence de notre database :

arguments > database_origin

Nom	Modifié le	Type	Taille
test	15/03/2023 11:22	Dossier de fichiers	
train	15/03/2023 11:22	Dossier de fichiers	

Le dossier database_origin est séparé en deux sous-dossiers, test et train. La répartition qui donne les meilleurs résultats pour la classification et qui est utilisée classiquement est une répartition 70/30.

Les dossiers friche et dépôts du test contiennent respectivement 154 et 153 images.

arguments > database_origin > test

Nom	Modifié le	Type	Taille
depots	15/03/2023 11:23	Dossier de fichiers	
friches	15/03/2023 11:24	Dossier de fichiers	

Les dossiers friche et dépôt du train contiennent respectivement 361 et 370 images.

Nom	Modifié le	Type	Taille
depots	15/03/2023 11:23	Dossier de fichiers	
friches	15/03/2023 11:24	Dossier de fichiers	

Entraînement d'un CNN : dissection

Convolution

Lors de l'entraînement de notre CNN, on va traiter nos images avec différents opérateurs : des couches de convolution, des couches de pooling, des couches de correction et des couches fully connected. En numérique, l'opération de convolution est matérialisée par un filtre kernel qui parcourt l'image de manière horizontale ou verticale pour faire ressortir certains détails. Ce mode de fonctionnement repose sur un axiome central : **les pixels voisins sont fortement corrélés !** Cette corrélation, qui découle du fait que les images représentent des formes qui ont des patrons qui se ressemblent quand ils représentent le même objet, nous permettent d'extraire des informations de l'image à l'aide de ces filtres, modélisés dans notre cas par des opérateurs de convolution.

Faisons passer une image issue de `database_origin\test\depots` pour mieux comprendre de quoi il en retourne.

```
# Définir le chemin du dossier contenant l'image
img_folder = r'C:\Users\gaspa\Downloads\database_origin\test\depots'

# Charger l'image à l'aide de la bibliothèque Pillow
img_path = os.path.join(img_folder, 'depot12.png')
img = tf.keras.preprocessing.image.load_img(img_path, target_size=(200, 200))

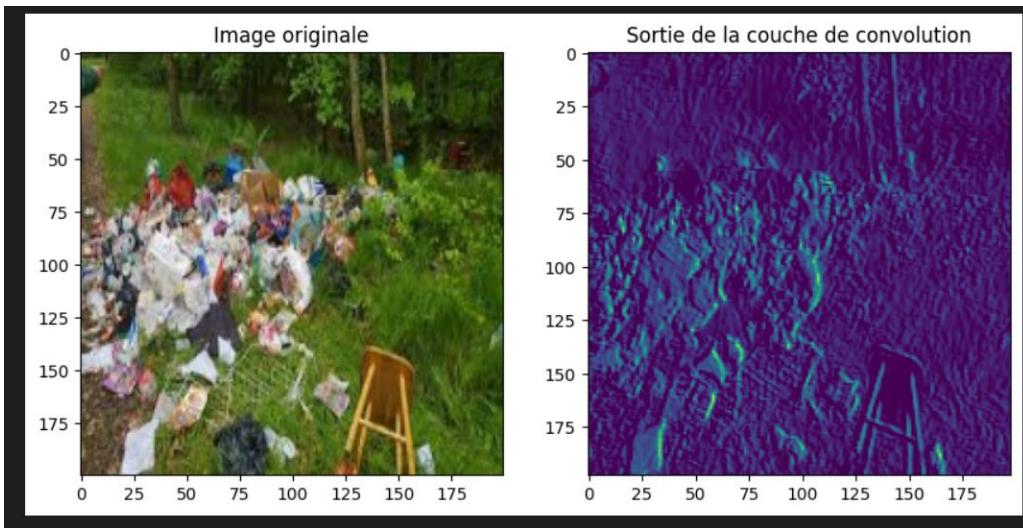
# Convertir l'image en un tableau numpy
img_array = tf.keras.preprocessing.image.img_to_array(img)

# Ajouter une dimension pour représenter le lot
img_array = np.expand_dims(img_array, axis=0)

# Définir une couche de convolution simple avec un noyau de 3x3 et 32 filtres
conv_layer = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3))

# Faire passer l'image par la couche de convolution
conv_output = conv_layer(img_array)

# Afficher l'image originale et la sortie de la couche de convolution
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
axs[0].imshow(img)
axs[0].set_title('Image originale')
axs[1].imshow(conv_output[0, :, :, 0])
axs[1].set_title('Sortie de la couche de convolution')
plt.show()
```



Ci-dessus, on a programmé un algorithme en python à l'aide de la librairie tensorflow et de keras pour qu'il nous renvoie en sortie l'image après son passage par une couche de convolution. Comme on s'y attendait, les images perdent en qualité. On remarque aussi que les features les plus discriminants ressortent : on voit bien que les détails qui aident le plus à l'identification de l'image sont augmentés pour être grossièrement visibles, tandis que les détails qui correspondent à des features moins discriminants sont écrasés par la couche de convolution. En entrée, l'image est de dimension [200, 200, 3] puis elle passe par une couche de convolution de noyau 3*3 et de 32 filtres. Chaque filtre extrait des caractéristiques différentes de l'image, créant ainsi un **tenseur**, ou tableau de caractéristique, de dimension [198, 198, 32]. Ces 32 dimensions correspondent aux dimensions des tableaux qui correspondent aux features de notre entraînement.

Comme on peut le remarquer, la convolution a légèrement réduit la dimension de nos images : pourquoi ? En fait, la dimension de la couche de sortie est plus petite que la dimension de l'image d'entrée car pendant la convolution, le noyau est centré sur chaque pixel, de sorte que les pixels au bord de l'image ne peuvent pas être entièrement recouverts par le noyau.

Pour contrer cet effet de bord, on va utiliser la technique du PADDING. Concrètement, le padding va remplacer la valeur de tous les pixels du bord par 0.

Couche de pooling

Les couches de pooling sont placées entre deux couches de convolution et prennent les features map en entrée. L'opération de pooling consiste à diminuer la taille de l'image tout en conservant un maximum de détails dans les caractéristiques. Le max pooling va découper les images en petits carrés de taille égale, par exemple 2*2 ou 3*3 pixels. Le pooling va chercher la valeur maximale parmi les pixels du groupe, et remplacer la valeur de chaque pixel du groupe par ce maximum. Cette opération illustre bien un des principes clés du deep-learning : augmenter au maximum les performances du modèle de réseau neuronal

en altérant volontairement la qualité des images, tout en conservant au maximum l'intégrité de l'information, pour ne pas altérer les résultats.

Cette fois ci, on remplace la couche de convolution par une couche de pooling :

```
img_path = os.path.join(img_folder, 'depot12.png')
img = tf.keras.preprocessing.image.load_img(img_path, target_size=(200, 200))

# Convertir l'image en un tableau numpy
img_array = tf.keras.preprocessing.image.img_to_array(img)

# Ajouter une dimension pour représenter le lot
img_array = np.expand_dims(img_array, axis=0)

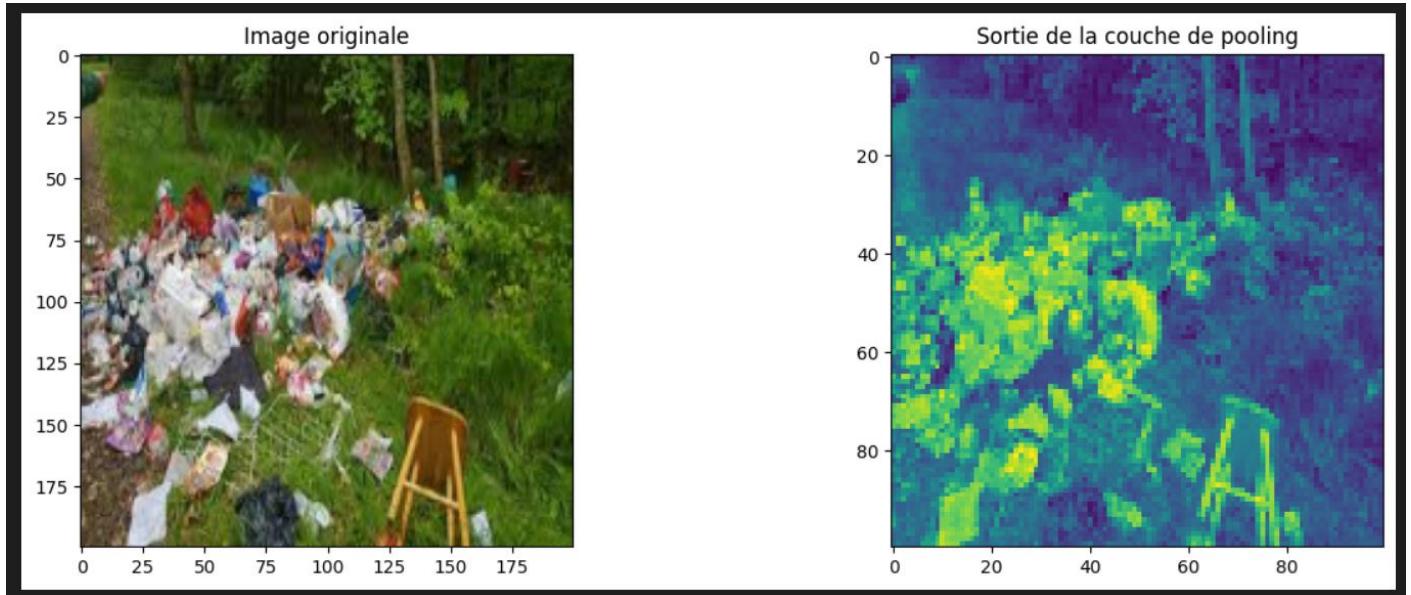
# Définir une couche de convolution simple avec un noyau de 3x3 et 32 filtres
conv_layer = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3))

# Faire passer l'image par la couche de convolution
conv_output = conv_layer(img_array)

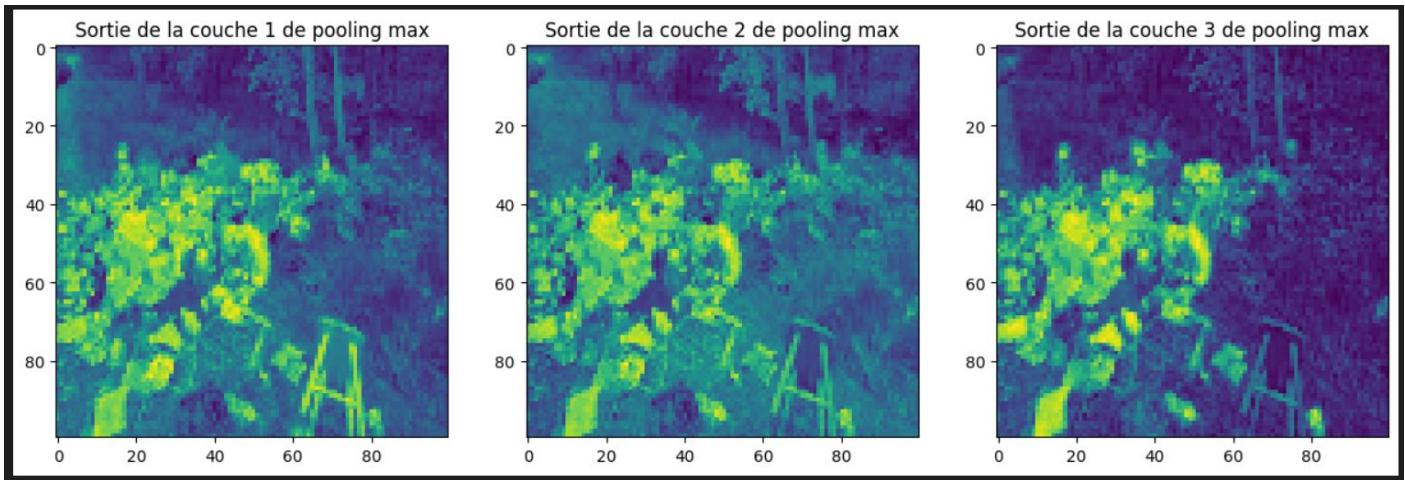
# Définir une couche de pooling avec un facteur de sous-échantillonnage de 2x2
pool_layer = tf.keras.layers.MaxPooling2D((2, 2))

# Faire passer la sortie de la couche de convolution par la couche de pooling
pool_output = pool_layer(conv_output)

# Afficher l'image originale, la sortie de la couche de convolution et la sortie de la couche de pooling
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
axs[0].imshow(img)
axs[0].set_title('Image originale')
#axs[1].imshow(conv_output[0, :, :, 0])
#axs[1].set_title('Sortie de la couche de convolution')
#axs[1].imshow(pool_output[0, :, :, 0])
#axs[1].set_title('Sortie de la couche de pooling')
```



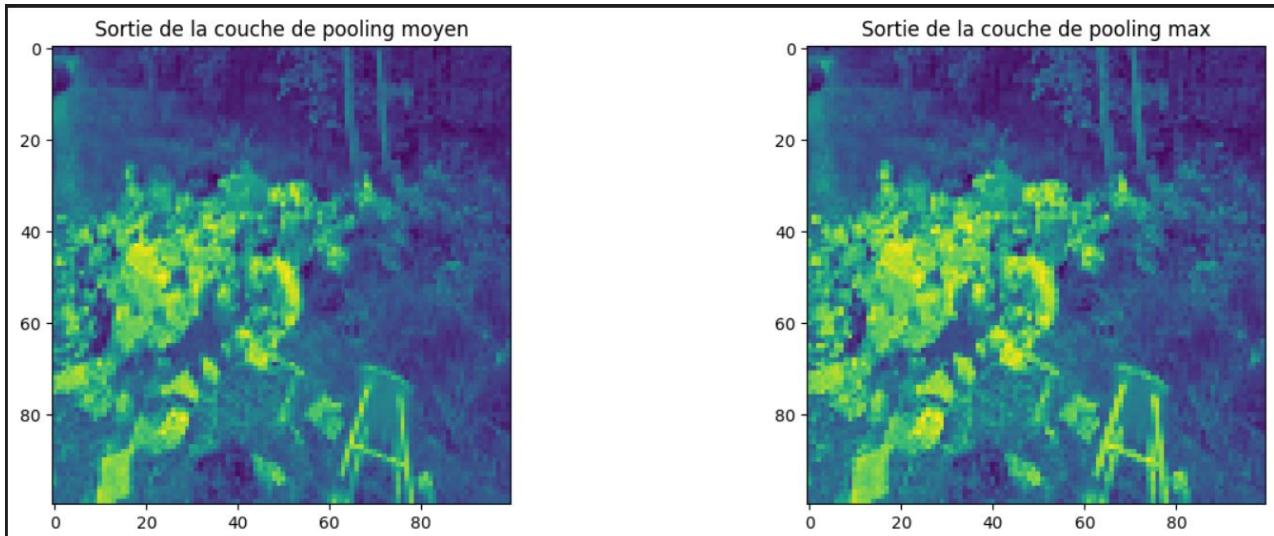
En sortie du pooling, on voit que la qualité est altérée, mais aussi que certaines caractéristiques ressortent clairement. Ces caractéristiques seront classées par le modèle à force de répétition sur les images du dataset : c'est bien la répétition, donc l'identification répétée de certaines caractéristiques sur les images du dataset, qui va permettre la solidité du modèle. On a choisi d'afficher la couche 0, mais il y en a bien 3 :



Pooling moyen versus Pooling Max

Le **pooling moyen**, implémentable sous python à travers le package keras de tensorflow, fait ressortir les features majoritaires de **manière plus douce**, sans effacer les features secondaires. C'est une méthode plus efficace si on a besoin de reconnaître des détails subtils.

Ci-dessous, voici la différence entre une couche de pooling moyen et une couche de pooling max sur une image de dépôt du dataset test.



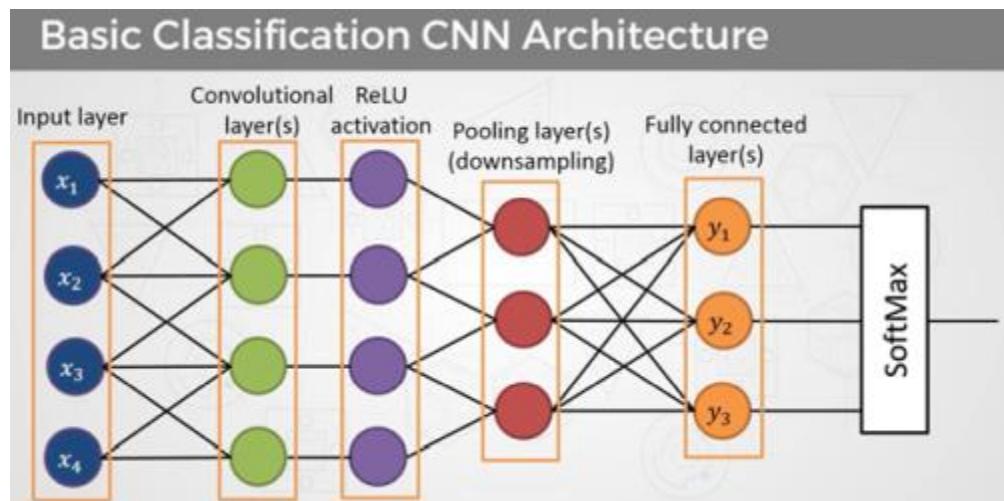
Couche d'aplatissement flatten layer

La couche d'aplatissement sert à aplatisir les données, c'est-à-dire qu'elle va prendre en entrée des données multidimensionnelles et va rendre en sortie des données unidimensionnelles. Pour nos données de tailles (200, 200, 3), la flatten layer va renvoyer un vecteur aplati de taille $200*200*3 = (120000,)$.

La couche d'aplatissement sert à préparer les données pour la couche dense qui traite les informations de façon unidimensionnelle.

Couche dense fully connected

Dans un CNN, la couche dense correspond à la couche où chaque neurone est connecté à tous les autres neurones de la couche précédente. Elle est complètement connectée car chaque neurone de la couche précédente est connecté à la couche dense. Elle est utilisée pour effectuer la classification.



Entraînement du CNN

```

# Configuration du générateur d'images pour la mise à l'échelle et l'augmentation des données
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# Génération des données d'entraînement à partir du dossier train_dir
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(200, 200),
    batch_size=4,
    class_mode='categorical',
    color_mode="rgb")

# Configuration du générateur d'images pour la mise à l'échelle des données de test
test_datagen = ImageDataGenerator(rescale=1./255)

# Génération des données de test à partir du dossier test_dir
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(200, 200),
    batch_size=10,
    class_mode='binary')

```

On réalise un premier entraînement en utilisant l'ImageDataGenerator pour faire de l'augmentation éphémère au cours de l'entraînement. On génère les données, puis on configure la taille du batch à 10.

On a utilisé une activation ReLu : pour les classifications binaires, il est vrai que la sigmoïde peut être particulièrement utile. Néanmoins, avec une plage de sortie limitée, on risquait une saturation qui aurait rendu l'apprentissage très long.

```

# Définition du modèle CNN
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), padding="SAME", activation='relu', input_shape=(200, 200, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compilation du modèle
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
              metrics=['accuracy'])

# Entraînement du modèle sur les données d'entraînement
model.fit(
    train_generator,
    steps_per_epoch=100,
    epochs=100,
    validation_data=test_generator,
    validation_steps=50)

```

✓ 63m 4.4s

Ensuite, on configure le CNN : on implémente 4 couches de convolution et 4 couches de pooling, 1 couche d'applatissement et 2 couches denses. On compile le modèle sur 100 epochs.

En sortie, malheureusement, on n'a pas les résultats espérés. On se rend compte que le modèle converge très vite vers une accuracy de 0.5. C'est un des signes qui permet d'identifier le surapprentissage. Ce surapprentissage peut être dû à un mauvais ajustement des hyperparamètres, mais dans notre cas, le problème vient du fait que la base de données est trop petite : pour entraîner un algorithme de deep learning convaincant, il faut des dizaines de milliers d'images, or nous n'en possédons que 1000.

```
Epoch 1/50
20/20 [=====] - ETA: 0s - loss: 0.6933 - accuracy: 0.5000WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can handle the number of samples required.
20/20 [=====] - 21s 936ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6939 - val_accuracy: 0.4272
Epoch 2/50
20/20 [=====] - 13s 619ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3/50
20/20 [=====] - 12s 615ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 4/50
20/20 [=====] - 13s 624ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 5/50
20/20 [=====] - 13s 612ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 6/50
20/20 [=====] - 12s 595ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 7/50
20/20 [=====] - 12s 616ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 8/50
20/20 [=====] - 13s 637ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 9/50
20/20 [=====] - 12s 604ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 10/50
20/20 [=====] - 13s 615ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 11/50
20/20 [=====] - 13s 647ms/step - loss: 0.6932 - accuracy: 0.5000
...
Epoch 49/50
20/20 [=====] - 13s 644ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 50/50
20/20 [=====] - 12s 612ms/step - loss: 0.6931 - accuracy: 0.5000
```

Pas de panique : régularisons

Méthodes de régularisation

En deep learning, les méthodes de régularisation sont utilisées lors de la compilation pour éviter le surapprentissage.

Sans le dire, on a déjà utilisé une méthode de régularisation : c'est l'augmentation de données, qui nous permettait d'augmenter notre volume sans altérer la véracité de notre BDD, pour aider le modèle à généraliser. Cette méthode n'a pas suffi à empêcher le surentraînement.

D'autres méthodes de régularisation s'offrent à nous.

Dropout

Le Dropout est une technique de régularisation qui permet d'éviter le surapprentissage. Lors d'un entraînement avec dropout, on désactive de manière aléatoire, selon une probabilité p , certaines couches du réseau neuronal. Le réseau s'entraîne donc en utilisant plusieurs combinaisons possibles de couches du réseau. Cette méthode permet une généralisation plus efficace du modèle.

Cette fois ci, on utilise l'average pooling et le dropout, et on lance l'entraînement :

```
# Définition du modèle CNN
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(200, 200, 3)),
    tf.keras.layers.AvgPool2D(pool_size=2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.AvgPool2D(pool_size=2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.AvgPool2D(pool_size=2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.AvgPool2D(pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

ou

```
Epoch 1/50
20/20 [=====] - ETA: 0s - loss: 0.6933 - accuracy: 0.5000WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or ge
20/20 [=====] - 21s 936ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6939 - val_accuracy: 0.4272
Epoch 2/50
20/20 [=====] - 13s 619ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3/50
20/20 [=====] - 12s 615ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 4/50
20/20 [=====] - 13s 624ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 5/50
20/20 [=====] - 13s 612ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 6/50
20/20 [=====] - 12s 595ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 7/50
20/20 [=====] - 12s 616ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 8/50
20/20 [=====] - 13s 637ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 9/50
20/20 [=====] - 12s 604ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 10/50
20/20 [=====] - 13s 615ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 11/50
20/20 [=====] - 13s 647ms/step - loss: 0.6932 - accuracy: 0.5000
...
Epoch 49/50
20/20 [=====] - 13s 644ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 50/50
20/20 [=====] - 12s 612ms/step - loss: 0.6931 - accuracy: 0.5000
```

Sur 50 epochs, on obtient là aussi une accuracy de 0.500, ce qui indique de nouveau un surapprentissage. Comme on est à court de données, on va combiner les deux méthodes de régularisations vues précédemment (augmentation et dropout) avec une nouvelle technique d'apprentissage : le transfer learning.

Régularisation par transfer learning

La technique du transfer learning consiste à réutiliser les poids d'un modèle entraîné et performant pour les injecter dans notre apprentissage. En partant d'un modèle pré entraîné plutôt que de zéro, on va permettre à notre algorithme d'apprentissage profond de se muscler avec moins de données

d'entraînement. Couramment utilisé dans les domaines du NLP, on va essayer de tester la pertinence du transfer learning sur notre dataset d'images.

Pour cela, on utilise le modèle pré-entraîné VGG16 de tensorflow.keras.applications.

La technique du transfer learning consiste à réutiliser les poids d'un modèle entraîné et performant pour les injecter dans notre apprentissage. En partant d'un modèle pré entraîné plutôt que de zéro, on va permettre à notre algorithme d'apprentissage profond de se muscler avec moins de données d'entraînement. Couramment utilisé dans les domaines du NLP, on va essayer de tester la pertinence du transfer learning sur notre dataset d'images.

Voici les éléments modifiés par rapport au code précédent :

```
# Chargement du modèle VGG16 pré-entraîné sans la dernière couche
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(200, 200, 3))

# Congeler toutes les couches du modèle VGG16 pré-entraîné pour éviter qu'elles ne soient mises à jour pendant l'entraînement
for layer in vgg.layers:
    layer.trainable = False

# Ajouter des couches personnalisées au-dessus du modèle VGG16 pré-entraîné
model = tf.keras.models.Sequential([
    vgg,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5), # Ajouter une couche Dropout pour éviter le surapprentissage
    tf.keras.layers.Dense(2, activation='softmax') # La dernière couche doit avoir autant de neurones que le nombre de classes dans le jeu de données personnalisé
])

# Compilation du modèle
model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
              metrics=['accuracy'])
```

Comme on peut le remarquer, on a chargé les poids du modèle pré-entraîné. On fait alors passer les images à travers deux couches denses et une couche flatten, avec un dropout de 0.5.

On a également implémenté une augmentation qui se fait au cours de la compilation.

On entraîne sur 50 epochs. Dès les premières epochs, on constate des scores très élevés en terme d'accuracy, avec une convergence rapide au-dessus de 90%. Ce succès laisse présager un très bon score ainsi qu'un algorithme de deep learning performant.

```
Epoch 1/50
100/100 [=====] - 226s 2s/step - loss: 0.5922 - accuracy: 0.7809 - val_loss: 0.5153 - val_accuracy: 0.8200
Epoch 2/50
100/100 [=====] - 222s 2s/step - loss: 0.3267 - accuracy: 0.8825 - val_loss: 0.3679 - val_accuracy: 0.8100
Epoch 3/50
100/100 [=====] - 232s 2s/step - loss: 0.3326 - accuracy: 0.8750 - val_loss: 0.6009 - val_accuracy: 0.7500
Epoch 4/50
100/100 [=====] - 232s 2s/step - loss: 0.2525 - accuracy: 0.9068 - val_loss: 0.3661 - val_accuracy: 0.8450
Epoch 5/50
100/100 [=====] - 225s 2s/step - loss: 0.2829 - accuracy: 0.8925 - val_loss: 0.3383 - val_accuracy: 0.8800
Epoch 6/50
100/100 [=====] - 167s 2s/step - loss: 0.2785 - accuracy: 0.8992 - val_loss: 0.3312 - val_accuracy: 0.8900
Epoch 7/50
100/100 [=====] - 208s 2s/step - loss: 0.2569 - accuracy: 0.9200 - val_loss: 0.2974 - val_accuracy: 0.8800
Epoch 8/50
100/100 [=====] - 228s 2s/step - loss: 0.2115 - accuracy: 0.9270 - val_loss: 0.2503 - val_accuracy: 0.9050
Epoch 9/50
100/100 [=====] - 201s 2s/step - loss: 0.1939 - accuracy: 0.9194 - val_loss: 0.3116 - val_accuracy: 0.8900
Epoch 10/50
100/100 [=====] - 179s 2s/step - loss: 0.2163 - accuracy: 0.9100 - val_loss: 0.3565 - val_accuracy: 0.8900
Epoch 11/50
100/100 [=====] - 184s 2s/step - loss: 0.2157 - accuracy: 0.9270 - val_loss: 0.3888 - val_accuracy: 0.8950
```

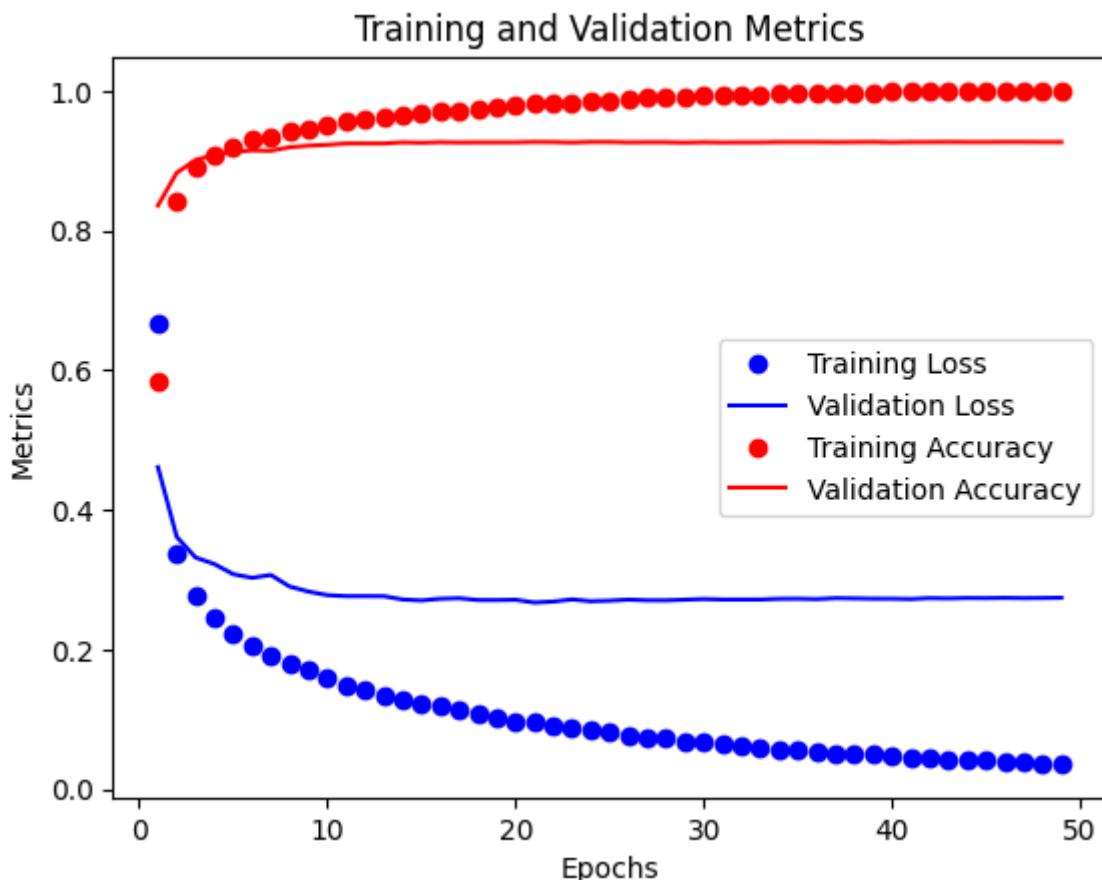
```

100/100 [=====] - 222s 2s/step - loss: 0.3267 - accuracy: 0.8825 - val_loss: 0.3679 - val_accuracy: 0.8100
Epoch 3/50
100/100 [=====] - 232s 2s/step - loss: 0.3326 - accuracy: 0.8750 - val_loss: 0.6009 - val_accuracy: 0.7500
Epoch 4/50
100/100 [=====] - 232s 2s/step - loss: 0.2525 - accuracy: 0.9068 - val_loss: 0.3661 - val_accuracy: 0.8450
Epoch 5/50
100/100 [=====] - 225s 2s/step - loss: 0.2829 - accuracy: 0.8925 - val_loss: 0.3383 - val_accuracy: 0.8800
Epoch 6/50
100/100 [=====] - 167s 2s/step - loss: 0.2785 - accuracy: 0.8992 - val_loss: 0.3312 - val_accuracy: 0.8900
Epoch 7/50
100/100 [=====] - 208s 2s/step - loss: 0.2569 - accuracy: 0.9200 - val_loss: 0.2974 - val_accuracy: 0.8800
Epoch 8/50
100/100 [=====] - 228s 2s/step - loss: 0.2115 - accuracy: 0.9270 - val_loss: 0.2503 - val_accuracy: 0.9050
Epoch 9/50
100/100 [=====] - 201s 2s/step - loss: 0.1939 - accuracy: 0.9194 - val_loss: 0.3116 - val_accuracy: 0.8900
Epoch 10/50
100/100 [=====] - 179s 2s/step - loss: 0.2163 - accuracy: 0.9100 - val_loss: 0.3565 - val_accuracy: 0.8900
Epoch 11/50
100/100 [=====] - 184s 2s/step - loss: 0.2157 - accuracy: 0.9270 - val_loss: 0.3888 - val_accuracy: 0.8950
Epoch 12/50
...
Epoch 49/50
100/100 [=====] - 225s 2s/step - loss: 0.0672 - accuracy: 0.9825 - val_loss: 0.3350 - val_accuracy: 0.9150
Epoch 50/50
100/100 [=====] - 224s 2s/step - loss: 0.0305 - accuracy: 0.9924 - val_loss: 0.4215 - val_accuracy: 0.9000

```

Comment interpréter ces résultats ?

On trace un graphique en python avec matplotlib pour afficher les valeurs de ces quatre métriques :



Loss

La loss mesure l'erreur moyenne entre les prédictions et les valeurs réelles de l'ensemble des données d'entraînement. Plus on minimise la loss, plus nos prédictions se rapprochent de la réalité. Notre loss est de 0.0305, ce qui signifie que notre modèle a réussi à minimiser notre fonction de coût.

Accuracy

L'accuracy, ou précision, mesure la proportion de prédictions correctes sur l'ensemble des prédictions. Si l'accuracy est maximisée, le modèle aura de bonnes chances de prédire avec précisions des données inconnues. Notre accuracy finale est de 0.9924, ce qui signifie que le modèle a réussi à classer avec précision 99% des données de notre ensemble.

Val_loss

La métrique val_loss, ou perte de validation, mesure l'erreur moyenne entre les prédictions du modèle et les valeurs réelles de l'ensemble de données de validation. Elle permet d'évaluer la capacité du modèle à généraliser, et à prédire avec précision des données inconnues. Il faut la minimiser pour que le modèle puisse bien généraliser.

Dans notre cas, la perte de validation est de 0.3, ce qui signifie que le modèle a obtenu une perte plus élevée sur l'ensemble de validation que sur l'ensemble d'entraînement. Cet écart peut venir d'un surajustement pendant l'entraînement (overfitting).

Val_accuracy

La précision de validation val_accuracy mesure l'accuracy sur les prédictions de la validation. Il faut maximiser cette valeur pour que le modèle puisse prédire les bonnes classes sur des données inconnues. Dans notre cas, elle est de 0.9, ce qui est bon, même si le fait qu'elle soit légèrement inférieure à l'accuracy pourrait indiquer un risque de surentraînement.

Détection d'images en temps réel

Il faut maintenant valider la dernière étape. On voulait une détection temps réel avec des boîtes englobantes qui identifient les classes de nos images sur une vidéo. On utilise l'apowersoft recorder et on prend une vidéo d'une vingtaine d'images de friches et de dépôt, qui sont étrangères aux images de la base de données.

On charge le modèle qu'on a entraîné et on opère une classification en bouclant sur les images de la vidéo.

```

import cv2
import tensorflow as tf
import numpy as np

# Charger le modèle CNN pré-entraîné
model = model

# Définir les classes que le modèle peut reconnaître
classes = ["depots", "friches"]

# Définir les couleurs pour les boîtes englobantes et les labels
colors = [(0, 255, 0), (0, 0, 255)]

# Charger la vidéo
cap = cv2.VideoCapture(r'C:\Users\gaspa\Downloads\ressources\video_test.mp4')

# Récupérer les dimensions de la vidéo
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Créer le fichier vidéo de sortie
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))

# Boucle sur les images de la vidéo
while(cap.isOpened()):
    # Lire l'image suivante

```

```

    # Redimensionner l'image pour qu'elle corresponde aux dimensions d'entrée du modèle
    input_image = cv2.resize(frame, (200, 200))

    # Prétraiter l'image pour la passer en entrée du modèle
    input_image = input_image.astype(np.float32) / 255.0
    input_image = np.expand_dims(input_image, axis=0)

    # Faire une prédiction avec le modèle
    predictions = model.predict(input_image)[0]

    # Trouver la classe la plus probable pour chaque prédiction
    class_idx = np.argmax(predictions)
    class_prob = predictions[class_idx]
    class_label = classes[class_idx]

    # Dessiner une boîte englobante autour de l'objet détecté
    color = colors[class_idx]
    cv2.rectangle(frame, (0, 0), (200, 30), (255,255,255), -1)
    cv2.putText(frame, f"{class_label}: {class_prob:.2f}", (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
    cv2.rectangle(frame, (0, 0), (width, height), color, 2)

    # Écrire l'image avec les boîtes englobantes dans le fichier vidéo de sortie
    out.write(frame)

# Fermer la vidéo et le fichier de sortie
cap.release()
out.release()
cv2

```

La vidéo est ensuite enregistrée automatiquement dans le dossier qui contient le code, au format avi.



Les friches détectées sont encadrées en rouge tandis que les dépôts d'ordures détectés sont encadrés en vert. Le nom de la classe apparaît, suivi d'une prédiction. Si le modèle ne détecte aucune des deux classes,

aucun cadre n'apparaîtra. En comptant le nombre de bonnes prédictions sur cette vidéo, on obtient 15 prédictions correctes pour 19 images, **soit un résultat de 78% de bonnes prédictions !**

Comme on peut le remarquer, les boîtes englobantes englobent toute l'image. On a d'abord pensé qu'on avait mal paramétré leur position, mais non ! En fait, les données d'entraînement de dépôts sont toujours situées dans la nature dans un paysage très similaire. L'algorithme de deep learning a donc enregistré la présence de cette nature comme un marqueur de la classe dépôt ! On lui a fait apprendre que dépôt signifie dépôt+ nature. Pour régler ce problème, on peut faire une base de données contenant des dépôts d'ordure urbains, dans la nature, etc. De cette manière, l'algorithme de DL fera une dichotomie entre le paysage et le dépôt d'ordures, et sera capable d'identifier précisément le dépôt.

Conclusion

On a donc réussi, sans base de données, avec un algorithme de scrapping utilisant selenium, à collecter 500 images de deux types sur internet. Grâce aux techniques d'augmentation du framework imgaug, on a porté ce nombre d'images à 1000 tout en conservant une qualité et une véracité dans la base de données.

Après quelques traitements des images, on a implémenté un réseau de neurones convolutionnel et combiné les méthodes de régularisation de l'augmentation, du dropout et du transfer learning, en ajustant le pooling, pour obtenir un modèle avec des scores étonnamment élevés d'accuracy et de loss, bien que les scores de val_loss et de val accuracy laissaient présager un léger surentraînement.

A l'aide de la librairie CV2, on a pu identifier, en direct, les classes « friches » et « dépôts » sur une vidéo, avec une précision de 78%.

On a donc montré qu'on peut aujourd'hui, avec très peu de ressources et beaucoup de techniques, réaliser de manière artisanale des algorithmes de détection d'image très performant.

Modeling

Les SVM et les méthodes de machine learning sont moins efficaces que les méthodes de deep learning dans la détection d'images.

Il existe des framework qu'on peut exécuter en python pour effectuer des opérations avec des réseaux neuronaux.

TensorFlow est un cadre d'apprentissage profond bien établi, et Keras est son API officielle de haut niveau qui simplifie la création de modèles. TensorFlow gère bien la reconnaissance et la classification.

La reconnaissance d'images consiste à introduire une image dans un réseau neuronal et à lui faire produire une sorte de label pour cette image. Le label produit par le réseau correspondra à une classe prédéfinie. L'image peut être étiquetée selon plusieurs classes ou selon une seule. S'il n'y a qu'une seule classe, on utilise souvent le terme "reconnaissance", alors qu'une tâche de reconnaissance multi-classes est souvent appelée "classification".

Un sous-ensemble de la classification d'images est la détection d'objets, où des instances spécifiques d'objets sont identifiées comme appartenant à une certaine classe, comme les animaux, les voitures ou les personnes.

La première couche d'un réseau neuronal prend en compte tous les pixels d'une image. Une fois que toutes les données ont été introduites dans le réseau, différents filtres sont appliqués à l'image, ce qui permet de représenter les différentes parties de l'image. Il s'agit de l'extraction de caractéristiques et de la création de "cartes de caractéristiques".

Ce processus d'extraction de caractéristiques d'une image est réalisé à l'aide d'une "couche convective", et la convolution consiste simplement à former une représentation d'une partie de l'image. C'est à partir de ce concept de convolution que nous obtenons le terme de réseau neuronal convolutif (CNN), le type de réseau neuronal le plus couramment utilisé dans la classification/reconnaissance d'images

En Deep Learning, les réseaux neuronaux convolutifs sont constitués de plusieurs couches et sont principalement utilisés pour le traitement d'images et la détection d'objets.

Les CNN sont largement utilisés pour identifier des images satellites, traiter des images médicales, prévoir des séries chronologiques et détecter des anomalies.

Tiny Yolo

Parmi les différents algorithmes qui existent, YOLO, pour « You Only Look Once », se démarque par ses performances. Nous pensons que son utilisation est accessible à notre niveau. Une fois la base de données parfaitement complétée, on entraînera un réseau de neurones Tiny Yolo.

Bibliographie

Bibliothèques indispensables python :

<https://selenium-python.readthedocs.io/>

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

<https://pypi.org/project/torch/>

<https://pytorch.org/vision/stable/transforms.html>

Pour en savoir plus sur la reconnaissance d'image :

<https://graphiste.com/blog/outils-reconnaissance-image/>

Un très bon blog sur la notion de reconnaissance d'image avec le DL :

<https://www.saagie.com/fr/quest-ce-que-la-detection-dobjet/>

A propos de la transformation des images pour l'augmentation de données :

<https://neptune.ai/blog/data-augmentation-in-python#:~:text=Data%20augmentation%20techniques&text=For%20example%2C%20for%20images%2C%20we,sharpen%20or%20blur%20an%20image>

Pourquoi Google nous met des bâtons dans les roues pour le scraping :

<https://www.webrankinfo.com/dossiers/google-search/denoncer-le-scraping>

Une démonstration des avantages de Tiny Tolo pour la détection d'images vidéo :

<https://medium.com/analytics-vidhya/yolov4-vs-yolov4-tiny-97932b6ec8ec>

Lien vers le github :

<https://github.com/Gaspard22082>