



**UFR des Sciences et Technologies**

**Master 2 des signaux et image en médecine**

**Rapport du projet**

# **Koala Instance Segmentation using Mask R-CNN**

**Réalisé par : AKIF Hajar**

**Encadré par : Mme MAUGARS Delphine**

**Année Universitaire : 2024-2025**



## Introduction :

Dans le cadre de ce projet, l'objectif était de développer un modèle performant capable de détecter et de segmenter les koalas dans des images avec précision. Pour cela, une approche basée sur la segmentation d'instance a été adoptée, en utilisant le modèle Mask R-CNN avec un backbone ResNet50. Ce choix repose sur la capacité de Mask R-CNN à combiner efficacement la détection d'objets et la segmentation, permettant non seulement d'identifier la présence des koalas mais aussi de délimiter précisément leurs contours dans diverses scènes.

## 1. Création de la dataset :

### 1.1. Choix :

Pour la création de la dataset, j'ai choisi d'utiliser des images de koalas afin de tester la segmentation d'objets sur un animal à la texture complexe et aux contours variables. Les images sont annotées avec des masques de segmentation précis, en utilisant des outils d'annotation manuelle pour s'assurer que chaque koalas est segmenté avec exactitude.

**Roboflow pour l'annotation** : J'ai utilisé **Roboflow**, une plateforme populaire pour l'annotation et la gestion de datasets. Cette plateforme facilite le processus d'annotation en offrant des outils de dessin précis et des formats d'exportation adaptés à divers frameworks, y compris COCO.

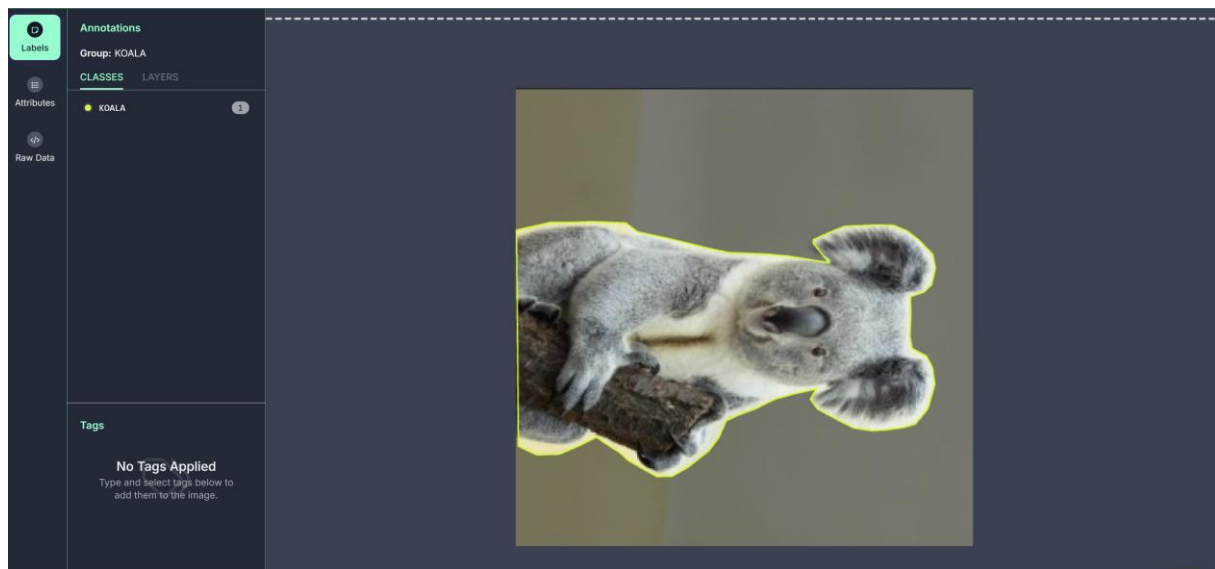
### 1.2. Méthodes utilisées :

#### a) Collecte des images :

J'ai collecté des images de léopards à partir de sources publiques disponibles en ligne, couvrant différentes poses, environnements et conditions d'éclairage.

#### b) Annotation des images avec Roboflow :

J'ai utilisé Roboflow pour annoter chaque image de manière précise, en traçant des polygones autour du léopard à l'aide de points de segmentation. Cette méthode a permis de délimiter avec précision les contours de l'animal, créant ainsi un masque de haute qualité. L'annotation point par point garantit une segmentation détaillée, particulièrement efficace pour capturer les motifs complexes du léopard et assurer une représentation fidèle de l'animal dans chaque image. (Voir les images).



#### c) Conversion au format COCO :

Après avoir annoté les images dans Roboflow, j'ai exporté les annotations au format **COCO**. Le format COCO est un standard pour les annotations en vision par ordinateur, et il contient à la fois les informations sur les **masques de segmentation** et les **bounding boxes** des objets annotés.

### 1.3. Dispatch de la dataset :

La dataset a été soigneusement répartie en trois ensembles distincts afin de maximiser les performances du modèle pendant l'entraînement et l'évaluation. La répartition des données s'est effectuée comme suit :

- Train Set (Ensemble d'entraînement) :
  - 87 % des images (soit 250 images) ont été utilisées pour l'entraînement du modèle.
  - Cet ensemble est destiné à permettre au modèle d'apprendre les caractéristiques des léopards et d'optimiser ses paramètres pour effectuer des prédictions de segmentation.
- Valid Set (Ensemble de validation) :
  - 8 % des images (soit 22 images) ont été utilisées pour la validation du modèle pendant l'entraînement.
  - Ces images ont permis de suivre les performances du modèle sur des données non vues pendant l'entraînement, afin de détecter d'éventuels sur-apprentissages (overfitting) et de régler les hyperparamètres en conséquence.
- Test Set (Ensemble de test) :
  - 5 % des images (soit 15 images) ont été mises de côté pour le test final du modèle.
  - Cet ensemble a été utilisé pour évaluer de manière objective les performances du modèle une fois l'entraînement terminé. Les résultats obtenus sur cet ensemble n'ont pas été utilisés pour ajuster le modèle pendant la phase d'entraînement.

## 2. Mise en place de Mask-RCNN avec TensorFlow 2.14


### 2.1. Clonage du dépôt GitHub :



The screenshot shows a Jupyter Notebook titled "Projet Segmentation Leopard.ipynb". The code cell contains the command `!git clone https://github.com/z-mahmud22/Mask-RCNN_TF2.14.0.git`. The output shows the progress of cloning the repository, including enumerating, counting, and compressing objects, and receiving the final files.

Cette commande télécharge l'intégralité des fichiers nécessaires pour le projet Mask-RCNN, qui est adapté à TensorFlow 2.14.0.

### 2.2. Navigation vers le répertoire du projet :



The screenshot shows a terminal window with the command `cd Mask-RCNN_TF2.14.0` and its output `/content/Mask-RCNN_TF2.14.0`.

Cela permet de préparer l'environnement pour exécuter les scripts du projet.

### 2.3. Configuration des librairies nécessaires :

Pour exécuter le projet, il est essentiel d'installer les librairies mentionnées dans le fichier de configuration des dépendances (requirements.txt).

```

25 | pip install -r requirements.txt
Requirement already satisfied: cython>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.1->-r requirements.txt (line 8)) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.1->-r requirements.txt (line 8)) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.1->-r requirements.txt (line 8)) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.1->-r requirements.txt (line 8)) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.1->-r requirements.txt (line 8)) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.1->-r requirements.txt (line 8)) (2.8.2)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.19.3->-r requirements.txt (line 13)) (3.4.2)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.19.3->-r requirements.txt (line 13)) (2024.9.20)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.19.3->-r requirements.txt (line 13)) (1.7.0)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (1.4.0)
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (1.68.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (3.7)
Requirement already satisfied: protobuf>=3.19.6 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (4.25.5)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (2.32.3)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard==2.14.1->-r requirements.txt (line 15)) (0.7.2)

```

Voici les principales bibliothèques et leurs versions utilisées dans le projet :

- **Cython=3.0.5** : Outil pour optimiser les performances des algorithmes Python en compilant du code C/C++.
- **H5py=3.9.0** : Permet de lire et écrire des fichiers HDF5, souvent utilisés pour sauvegarder les modèles de Deep Learning.
- **Imgaug=0.4.0** : Bibliothèque de traitement d'images pour augmenter les datasets en appliquant des transformations (rotation, zoom, etc.).
- **IPython=7.34.0** : Environnement interactif pour exécuter des scripts Python de manière dynamique.
- **IPython-genutils=0.2.0** : Fournit des outils utilitaires pour IPython et Jupyter.
- **IPython-sql=0.5.0** : Intègre des commandes SQL directement dans les notebooks Jupyter pour manipuler les bases de données.
- **Keras=2.14.0** : API simplifiée pour concevoir, entraîner et évaluer des modèles d'apprentissage profond.
- **Matplotlib=3.7.1** : Outil de visualisation pour tracer des graphiques, des courbes et des images issues des traitements.
- **Numpy=1.23.5** : Bibliothèque fondamentale pour les calculs numériques et la manipulation d'arrays multidimensionnels.
- **Opencv-python=4.8.0.76** : Implémente des algorithmes pour le traitement d'images et la vision par ordinateur.
- **Opencv-contrib-python=4.8.0.76** : Version enrichie d'Opencv avec des fonctionnalités supplémentaires (e.g., algorithmes expérimentaux).
- **Pillow=9.4.0** : Manipulation d'images (redimensionnement, conversion de formats, etc.).
- **Scikit-image=0.19.3** : Fournit des outils pour l'analyse et la transformation d'images (segmentation, filtrage, etc.).
- **Scipy=1.11.3** : Complément de NumPy pour les calculs scientifiques avancés (statistiques, intégration, optimisation).
- **Tensorboard=2.14.1** : Outil de visualisation et de suivi des métriques d'entraînement des modèles TensorFlow.
- **Tensorflow[and-cuda]=2.14.0** : Bibliothèque centrale pour construire et entraîner des modèles de Deep Learning. Intègre le support CUDA pour exploiter les GPU NVIDIA.

## 2.4. Bibliothèques Importées et Leur Utilisation dans le Projet :

### • Manipulation et analyse des données :

Numpy : Calculs numériques avancés, manipulation de matrices et de tenseurs.

Pandas : Manipulation et analyse de données tabulaires, comme les DataFrames.

Scipy.stats : Calcul de statistiques avancées (tests d'hypothèses, distributions, etc.).

- **Visualisation des données :**

Matplotlib.pyplot : Traçage de graphiques statiques (histogrammes, courbes, etc.).

Seaborn : Visualisations statistiques basées sur Matplotlib, avec des graphiques améliorés.

Plotly.express et Plotly.graph\_objects : Outils interactifs pour des visualisations avancées.

- **Apprentissage profond et vision par ordinateur**

Tensorflow : Framework d'apprentissage profond pour la construction de réseaux neuronaux.

Keras : API haut niveau intégrée à TensorFlow pour concevoir des modèles d'IA.

keras.backend : Accès à des fonctions bas niveau comme les opérations sur les tenseurs.

Mrcnn : Implémentation de Mask R-CNN pour la segmentation d'objets dans les images.

Mrcnn.config : Définit les configurations de l'entraînement.

Mrcnn.model.MaskRCNN : Modèle principal de segmentation d'objets.

Mrcnn.utils.Dataset : Classe pour manipuler les datasets.

Mrcnn.visualize : Visualisation des prédictions et annotations.

- **Traitement et augmentation des images**

Cv2 : OpenCV, bibliothèque pour le traitement d'images et la vision par ordinateur.

Imgaug : Augmentation de données pour l'apprentissage automatique (flips, rotations, etc.).

Albumentations : Librairie avancée pour l'augmentation d'images.

skimage.io et skimage.transform : Chargement et transformation d'images.

- **Préparation des données :**

sklearn.model\_selection.train\_test\_split : Séparation des datasets en ensembles d'entraînement et de test.

sklearn.preprocessing.StandardScaler : Normalisation des données pour optimiser les modèles.

sklearn.metrics : Évaluation des performances des modèles (matrice de confusion, MSE).

- **Gestion des fichiers et des systèmes :**

Os : Gestion des chemins, dossiers et fichiers locaux.

Json : Manipulation des fichiers JSON pour gérer des annotations ou configurations.

Sys : Interactions avec l'interpréteur Python et manipulation des arguments.

Time : Mesure des performances ou temporisation.

- **Visualisation des annotations et gestion des datasets**

pycocotools.coco : Travail avec le format COCO pour les annotations et datasets.

H5py : Lecture et écriture de fichiers HDF5 pour sauvegarder des modèles ou données volumineuses.

PIL.Image : Manipulation des images (redimensionnement, rotation, conversion).

- **Autres outils utilitaires :**

Random : Génération de nombres aléatoires pour des tâches comme l'échantillonnage.

Tqdm : Barre de progression pour suivre l'exécution des boucles.

```

import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import tensorflow as tf
import keras
from keras import backend as K
import mrcnn
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import Dataset
import cv2
import imgaug
import albumentations as A
from skimage import io, transform
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    mean_squared_error
)
import os
import json
import sys
import time
import random
from tqdm import tqdm
import pycocotools.coco as coco
import h5py
from PIL import Image

```

### 3. Pipeline Complet pour la Segmentation d'Images avec Mask R-CNN :

#### 3.1. Explication de la procédure pour accéder aux données stockées sur Google Drive :

Accès aux fichiers via l'API Google Drive.

Monter Google Drive dans Colab en utilisant le code suivant :

```

from google.colab import drive
drive.mount('/content/drive')

```

Mounted at /content/drive

Le répertoire /content/drive/MyDrive/KOALA.v1i.coco/ contient l'ensemble des données requises pour l'entraînement du modèle Mask R-CNN, incluant les sous-dossiers train, val et test.

```

[ ] import os
from pycocotools.coco import COCO

# Specify the path to your dataset (change the path accordingly)
data_dir = '/content/drive/MyDrive/KOALA.v1i.coco/'

```

#### 3.2. Chargement du Dataset "Koala" pour Mask R-CNN Load instance masks for an image.

```

import os
import logging
from typing import List, Tuple, Optional

import numpy as np
from pycocotools.coco import COCO
from mrcnn.utils import Dataset

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class KOALADataset(Dataset):
    def load_KOALA(self, dataset_dir: str, subset: str) -> None:
        """
        Load a subset of the KOALA dataset.

```



```
Args:
    dataset_dir: Root directory of the dataset.
    subset: The subset to load: 'train', 'valid', or 'test'.

Raises:
    FileNotFoundError: If dataset directory or annotations file
is missing.
    ValueError: If subset is not valid.
"""
# Validate subset
valid_subsets = ['train', 'valid', 'test']
if subset not in valid_subsets:
    raise ValueError(f"Invalid subset. Must be one of
{valid_subsets}")

# Add class
self.add_class("KOALA", 1, "KOALA")

# Validate dataset directory
if not os.path.exists(dataset_dir):
    raise FileNotFoundError(f"Dataset directory not found:
{dataset_dir}")

# Define the path to the annotation file for the current subset
annotations_path = os.path.join(dataset_dir, subset,
'_annotations.coco.json')

if not os.path.exists(annotations_path):
    raise FileNotFoundError(f"Annotations file not found:
{annotations_path}")

try:
    # Load the COCO annotations for the current subset
    coco = COCO(annotations_path)

    # Add images to the dataset
    image_ids = coco.getImgIds()
    logger.info(f>Loading {len(image_ids)} images from {subset}
subset")

    for image_id in image_ids:
        image_info = coco.loadImgs(image_id)[0]
        image_path = os.path.join(dataset_dir, subset,
image_info['file_name'])

        if not os.path.exists(image_path):
```

```

        logger.warning(f"Image not found: {image_path}")
        continue

    self.add_image(
        "KOALA",
        image_id=image_id,
        path=image_path,
        width=image_info['width'],
        height=image_info['height']
    )

except Exception as e:
    logger.error(f"Error loading KOALA dataset: {e}")
    raise

def load_mask(self, image_id: int) -> Tuple[np.ndarray,
np.ndarray]:
    """
    Load instance masks for an image.

    Args:
        image_id: Unique identifier for the image.

    Returns:
        A tuple of masks (numpy array) and corresponding class IDs.
    """
    try:
        info = self.image_info[image_id]
        annotations_path =
os.path.join(os.path.dirname(info['path']), '_annotations.coco.json')

        coco = COCO(annotations_path)
        ann_ids = coco.getAnnIds(imgIds=info['id'])
        annotations = coco.loadAnns(ann_ids)

        masks = []
        class_ids = []
        for ann in annotations:
            mask = coco.annToMask(ann)
            masks.append(mask)
            class_ids.append(self.map_category_id(ann['category_id']
)))

        # Handle empty masks scenario
        if not masks:
            masks = np.zeros((info['height'], info['width'], 0),
dtype=np.uint8)

```

```

        class_ids = np.array([], dtype=np.int32)
    else:
        masks = np.stack(masks, axis=-1)
        class_ids = np.array(class_ids)

    return masks, class_ids

except Exception as e:
    logger.error(f"Error loading mask for image {image_id}: {e}")
    raise

def map_category_id(self, original_id: int) -> int:
    """
    Map original category IDs to dataset-specific class IDs.

    Args:
        original_id: Category ID from original dataset.

    Returns:
        Mapped class ID for the dataset.
    """
    # Example mapping - adjust based on your specific dataset
    return 1 # Default to KOALA class

def image_reference(self, image_id: int) -> str:
    """
    Get the file path for a given image ID.

    Args:
        image_id: Unique identifier for the image.

    Returns:
        Absolute file path of the image.
    """
    return self.image_info[image_id]['path']

```

- **Résumé :** Le code présenté permet de charger un dataset au format COCO pour entraîner un modèle Mask R-CNN. Il gère le chargement des images et de leurs annotations, ainsi que la génération de masques binaires pour la segmentation d'objets spécifiques. Pour assurer le bon fonctionnement, il est essentiel de respecter une organisation rigoureuse du dataset avec des sous-dossiers train, valid et test, ainsi que des fichiers d'annotations au format COCO (\_annotations.coco.json).
- **Étape 1 : Définition de la classe KOALADataset**  
La classe KOALADataset hérite de la classe de base Dataset (fournie par Mask R-CNN) qui permet de gérer efficacement des datasets personnalisés pour des tâches de vision par ordinateur. Elle est configurée pour inclure des méthodes spécifiques au dataset KOALA.
- **Étape 2 : Chargement du dataset "Koala" avec la méthode load\_KOALA :**

- ✓ Ajout d'une classe : `self.add_class ("Koala", 1, "Koala")` : **`self.add_class("KOALA", 1, "KOALA")`** : Ajoute la classe "KOALA" avec l'ID 1, nécessaire pour que Mask R-CNN identifie les objets correspondants dans les masques d'instance.
- ✓ Chargement des annotations COCO : Le chemin du fichier d'annotations (`_annotations.coco.json`) est construit à partir du répertoire de données et du sous-ensemble (train, valid, ou test). La bibliothèque COCO est utilisée pour charger le fichier d'annotations et accéder aux images.
- ✓ Ajout des images : `coco.getImgIds()` permet d'obtenir les IDs des images dans les annotations. Pour chaque image, le code charge les informations nécessaires (nom de fichier, largeur, hauteur) et ajoute cette image au dataset via `self.add_image`.
- **Étape 3 : Chargement des masques d'instance avec la méthode `load_mask` :**
  - ✓ Chargement des annotations pour une image spécifique :  
`info = self.image_info[image_id]` récupère les informations de l'image à partir de l'ID. Le fichier d'annotations est rechargé à partir du chemin d'accès spécifique à l'image (`_annotations.coco.json`).
  - ✓ Récupération des annotations pour l'image :  
`coco.getAnnIds(imgIds=info['id'])` récupère les annotations pour l'image spécifiée.  
`coco.loadAnns(ann_ids)` charge les annotations sous forme de listes.
  - ✓ Création des masques : Pour chaque annotation, `coco.annToMask(ann)` convertit l'annotation en un masque binaire. Les masques sont ajoutés à une liste et empilés dans un tableau 3D (une dimension pour chaque masque d'instance).
  - ✓ Traitement des masques vides : Si aucun masque n'est trouvé, un masque vide est créé (un tableau numpy rempli de zéros). Si des masques sont présents, ils sont empilés le long du dernier axe pour former une matrice 3D, et les masques vides sont filtrés.
- **Étape 4 : Retourner le chemin de l'image avec la méthode `image_reference` :** Cette méthode retourne le chemin du fichier image correspondant à l'ID spécifié, ce qui permet d'accéder directement à l'image à partir du dataset.

### 3.3. Entraînement d'un modèle Mask R-CNN pour la détection de koala à l'aide d'un dataset COCO :

#### a) Initialisation et Préparation des Datasets (Entraînement, Validation et Test) pour Mask R-CNN

```
# Initialize and load the training dataset
dataset_train = KOALADataset()
dataset_train.load_KOALA(data_dir, 'train')
dataset_train.prepare()

# Initialize and load the validation dataset
dataset_val = KOALADataset()
dataset_val.load_KOALA(data_dir, 'valid')
dataset_val.prepare()

# Initialize and load the testing dataset
dataset_test = KOALADataset()
dataset_test.load_KOALA(data_dir, 'test')
dataset_test.prepare()
```

- **Résumé :** Ce bloc de code initialise les datasets d'entraînement, de validation et de test, en les chargeant via la classe KOALADataset. Chaque dataset est préparé pour être utilisé dans un modèle Mask R-CNN, grâce à la méthode prepare qui organise les données pour un traitement optimal.
- **Initialisation et chargement du dataset d'entraînement :**
  - ✓ Création d'une instance dataset\_train de la classe KOALADataset.
  - ✓ Chargement des images et annotations du sous-ensemble train à partir du répertoire data\_dir.
  - ✓ Préparation des données pour structurer les informations nécessaires au modèle Mask R-CNN.
- **Initialisation et chargement du dataset de validation :**
  - ✓ Création d'une instance dataset\_val pour le dataset de validation.
  - ✓ Chargement des données du sous-ensemble valid depuis le répertoire spécifié.
  - ✓ Organisation des données pour évaluer les performances pendant l'entraînement.
- **Initialisation et chargement du dataset de test :**
  - ✓ Création d'une instance dataset\_test pour le dataset de test.
  - ✓ Chargement des données du sous-ensemble test, utilisé pour l'évaluation finale.
  - ✓ Préparation des données pour leur utilisation dans les tests finaux du modèle.

#### b) Configuration Personnalisée pour l'Entraînement d'un Modèle Mask R-CNN :

```
# Import necessary libraries
from mrcnn.model import MaskRCNN
from mrcnn.config import Config

# Define your custom configuration class
class CustomConfig(Config):
    """
    Configuration for training on a custom dataset.
    Adjust the parameters to fit your dataset and hardware.
    """
    NAME = "koala" # Name of the dataset or task
    IMAGES_PER_GPU = 2 # Adjust this based on the memory of your GPU
    NUM_CLASSES = 1 + 1 # Background (1) + koala (1)
    STEPS_PER_EPOCH = 100 # Number of training steps per epoch
    VALIDATION_STEPS = 50 # Number of validation steps at the end of
each epoch
    LEARNING_RATE = 0.001 # Initial learning rate
    IMAGE_MIN_DIM = 512 # Minimum image dimension for resizing
    IMAGE_MAX_DIM = 512 # Maximum image dimension for resizing
    GPU_COUNT = 1 # Number of GPUs to use

    # Add other optional parameters as needed, for example:
    # BACKBONE = "resnet50" # Backbone architecture (default:
resnet101)
    # RPN_ANCHOR_SCALES = (32, 64, 128, 256, 512) # Anchor box scales

# Instantiate your custom configuration
config = CustomConfig()

# Display configuration to verify settings
```

```
print("Custom configuration:")
for attribute, value in vars(config).items():
    print(f"{attribute}: {value}")
```

- **Résumé :** Ce code personnalise les paramètres nécessaires à l'entraînement d'un modèle Mask R-CNN sur la dataset koala. Les ajustements permettent d'optimiser les performances selon les caractéristiques du dataset et les contraintes matérielles.
- **Définition de la classe de configuration personnalisée:**

J'ai personnalisé la classe Config pour adapter Mask R-CNN au dataset des Koalas.

- ✓ NAME : Nom de la configuration pour identification (ici, "koala").
- ✓ IMAGES\_PER\_GPU: Nombre d'images traitées par GPU, ajusté en fonction de la mémoire disponible.
- ✓ NUM\_CLASSES : Total des classes (arrière-plan + classe "koala").
- ✓ STEPS\_PER\_EPOCH : Nombre de pas d'entraînement par époque.
- ✓ VALIDATION\_STEPS : Nombre de pas pour la validation en fin d'époque.
- ✓ LEARNING\_RATE: Taux d'apprentissage initial.
- ✓ IMAGE\_MIN\_DIM et IMAGE\_MAX\_DIM : Dimensions minimales et maximales pour le redimensionnement des images.
- ✓ GPU\_COUNT : Nombre de GPU utilisés pour l'entraînement.

### c) Étapes pour l'Initialisation, l'Entraînement et la Sauvegarde du Modèle Mask R-CNN :

```
# Step 1: Create the model in training mode
model = MaskRCNN(mode="training", config=config, model_dir='/content')

# Step 2: Load pre-trained weights (COCO weights) but exclude the
conflicting layers
model.load_weights('mask_rcnn_coco.h5', by_name=True,
exclude=["mrcnn_bbox_fc", "mrcnn_class_logits", "mrcnn_mask"])

# Step 3: Train the model (fine-tuning only the heads)
model.train(dataset_train, dataset_val,
learning_rate=config.LEARNING_RATE, epochs=10, layers='heads')

# Sauvegarde des poids
output_weights_path = os.path.join(model.model_dir,
"mask_rcnn_koala_bbox.h5")
model.keras_model.save_weights(output_weights_path)

print(f"Poids sauvegardés dans : {output_weights_path}")
```

- **Étape 1 : Création du modèle :** Pour commencer, j'ai créé une instance du modèle Mask R-CNN en mode training. J'ai utilisé une configuration personnalisée définie dans la classe CustomConfig pour ajuster les paramètres du modèle à mon dataset et à ma machine. J'ai également spécifié un répertoire de travail (model\_dir) pour stocker les logs et les poids générés pendant l'entraînement.
- **Étape 2 : Chargement des poids pré-entraînés :** J'ai utilisé les poids pré-entraînés sur le dataset COCO pour initialiser le modèle.

- **Étape 3 : Entraînement du modèle :** Pour l'entraînement, j'ai utilisé mon dataset personnalisé, divisé en sous-ensembles d'entraînement et de validation. Je n'ai entraîné que les couches supérieures du réseau (fine-tuning des heads), en fixant le taux d'apprentissage à une valeur adaptée (config.LEARNING\_RATE) et en définissant le nombre d'époques à 10. Cela m'a permis de concentrer l'apprentissage sur les caractéristiques spécifiques à mon dataset tout en conservant les représentations générales apprises sur COCO.
- **Étape 4 : Sauvegarde des poids :** Une fois l'entraînement terminé, j'ai sauvegardé les poids résultants dans un fichier .h5. Cette sauvegarde me permet de réutiliser ces poids pour tester le modèle ou poursuivre un nouvel entraînement à partir de cet état. J'ai pris soin de vérifier que les poids étaient bien enregistrés dans le répertoire spécifié.

#### d) Sauvegarde de l'historique d'entraînement et génération de graphique des pertes (losses)

```
import os
import matplotlib.pyplot as plt

# Sauvegarder l'historique de l'entraînement, y compris bbox loss
history_path = os.path.join('/content', 'training_logs.txt')
history = model.keras_model.history.history # Historique
d'entraînement

# Écrire les logs dans un fichier texte
with open(history_path, 'w') as log_file:
    for key, values in history.items():
        log_file.write(f"{key}: {values}\n")

print(f"Historique d'entraînement sauvegardé dans : {history_path}")

# Générer un graphique des pertes (losses)
epochs = range(1, len(history['loss']) + 1)

plt.figure(figsize=(10, 6))

# Tracer la loss totale
plt.plot(epochs, history['loss'], 'bo-', label='Training Loss')
plt.plot(epochs, history['val_loss'], 'r*-', label='Validation Loss')

# Tracer la bbox loss
if 'mrcnn_bbox_loss' in history:
    plt.plot(epochs, history['mrcnn_bbox_loss'], 'g^-', label='BBox
Training Loss')
if 'val_mrcnn_bbox_loss' in history:
    plt.plot(epochs, history['val_mrcnn_bbox_loss'], 'ys-', label='BBox
Validation Loss')

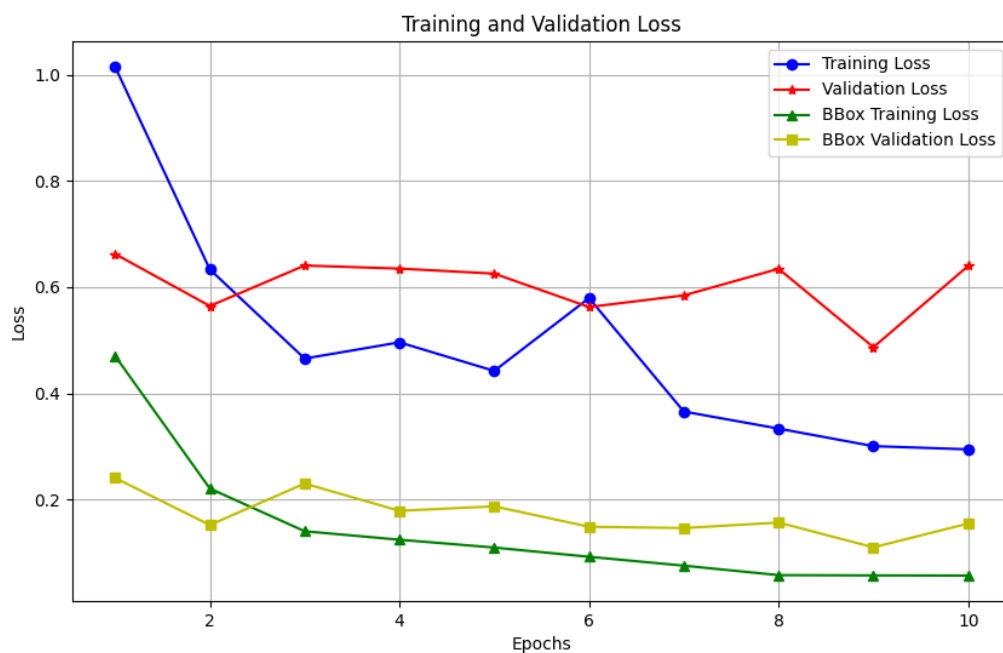
# Configurer le graphique
plt.title('Training and Validation Loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

# Sauvegarder le graphique
plot_path = os.path.join('/content', 'loss_plot.png')
plt.savefig(plot_path)
plt.show()

print(f"Graphique des pertes sauvegardé dans : {plot_path}")
```

- **Résumé :** Le script permet de sauvegarder l'historique des pertes de l'entraînement dans un fichier texte et génère un graphique illustrant l'évolution de la perte totale ainsi que de la perte de boîte englobante (si disponible) pendant l'entraînement et la validation. Les résultats sont ensuite sauvegardés sous forme de fichier PNG et affichés à l'utilisateur.
- **Analyse des Résultats :**



✓ **Commentaire :**

- **Training Loss :** La courbe bleue diminue régulièrement (1.0 → 0.4), montrant un apprentissage efficace du modèle.
- **Validation Loss :** La courbe rouge suit le Training Loss au début, indiquant un bon potentiel de généralisation.
- **BBox Training Loss :** La courbe verte reste basse et stable, reflétant une localisation précise des objets.
- **BBox Validation Loss :** La courbe jaune est proche de la BBox Training Loss, montrant une bonne généralisation sur les boîtes englobantes.



### e) Analyse des Performances du Modèle à l'aide de la Matrice de Confusion :

```
# Step: Generate Confusion Matrix after training
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Step: Create a model for inference
class InferenceConfig(Config):
    NAME = "KOALA"
    NUM_CLASSES = 1 + 1 # Background + koala
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

inference_config = InferenceConfig()
inference_model = MaskRCNN(mode="inference", config=inference_config,
model_dir='/content')

# Load the trained weights into the inference model
inference_model.load_weights(model.find_last(), by_name=True)

# Step 4.2: Collect true and predicted labels
true_labels = []
pred_labels = []

for image_id in dataset_test.image_ids:
    # Load true labels (class IDs) from the dataset
    true_masks, true_classes = dataset_test.load_mask(image_id)

    # Predict the image using the inference model
    image = dataset_test.load_image(image_id)
    pred = inference_model.detect([image], verbose=0)[0]

    # Align true and predicted classes
    if len(true_classes) == len(pred['class_ids']):
        true_labels.extend(true_classes)
        pred_labels.extend(pred['class_ids'])
    else:
        # Handle mismatched cases
        true_labels.extend(true_classes)
        pred_labels.extend(pred['class_ids'])

    # Fill with '0' (background class) for unmatched entries
    if len(true_classes) > len(pred['class_ids']):
        pred_labels.extend([0] * (len(true_classes) -
len(pred['class_ids'])))
    elif len(pred['class_ids']) > len(true_classes):
        true_labels.extend([0] * (len(pred['class_ids']) -
len(true_classes)))
```

```
# Step 4.3: Generate and plot the confusion matrix
cm = confusion_matrix(true_labels, pred_labels, labels=[0, 1]) # Add
'0' for background
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['background', 'KOALA'])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

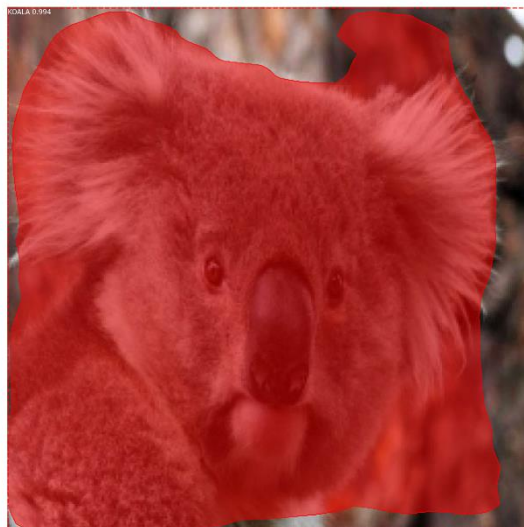
# Step 5: Visualize test predictions (Successes and Mistakes)
from mrcnn.visualize import display_instances
from mrcnn import visualize

import random

# Select 5 random image IDs from the test dataset
for i in random.sample(list(dataset_test.image_ids), 5):
    image = dataset_test.load_image(i) # Load the image
    results = inference_model.detect([image], verbose=0) # Perform
detection

    # Visualize the prediction
    visualize.display_instances(
        image,
        results[0]['rois'],
        results[0]['masks'],
        results[0]['class_ids'],
        dataset_test.class_names,
        results[0]['scores']
    )
```

- **Résumé** : J'évalue les performances de mon modèle à la fois quantitativement (grâce à la matrice de confusion) et qualitativement (via la visualisation des prédictions).
- **Génération de la Matrice de Confusion** : J'utilise les étiquettes vraies et prédites pour évaluer la performance de mon modèle. J'affiche une matrice de confusion pour comparer les classes détectées, notamment "background" et "koala".
- **Création d'un Modèle d'Inférence** : Je configure un modèle spécifiquement pour l'inférence (nom, classes, gestion du GPU). Je charge les poids déjà entraînés pour effectuer des prédictions sur des images inédites.
- **Collecte des Étiquettes** : Je compare les classes réelles et les classes prédites pour chaque image de l'ensemble de test. J'aligne les listes en ajoutant des classes de remplissage si les longueurs ne correspondent pas.
- **Visualisation des Résultats de Test** : Je sélectionne aléatoirement 5 images de mon ensemble de test. Je visualise les prédictions en montrant les régions détectées, les masques, les scores et les classes.



## f) Évaluation de la Performance d'un Modèle d'Instance Segmentation : Calcul de la Moyenne de la Précision Moyenne (mAP) :

```
from mrcnn.utils import compute_ap, extract_bboxes
import numpy as np
import cv2 # Assurez-vous que OpenCV est installé
from mrcnn import utils

# Liste pour stocker les AP (Average Precision) pour chaque image
APs = []

# Parcourir toutes les images du dataset de test
for image_id in dataset_test.image_ids:
    # Charger l'image
    image = dataset_test.load_image(image_id)

    # Charger les données au sol (ground truth): masques et IDs de
    classe
    gt_mask, gt_class_ids = dataset_test.load_mask(image_id)

    # Générer les boîtes englobantes au sol (à partir des masques si
    elles ne sont pas prédéfinies)
    gt_bbox = extract_bboxes(gt_mask)

    # Effectuer l'inférence sur l'image
    results = inference_model.detect([image], verbose=0)
    r = results[0]

    # Vérifier si des prédictions existent
    if r['masks'].size == 0:
        continue

    # Redimensionner les masques prédits pour correspondre aux masques
    au sol (si nécessaire)
    if r['masks'].shape[:2] != gt_mask.shape[:2]:
        pred_masks_resized = np.zeros(gt_mask.shape, dtype=np.uint8)
        for i in range(r['masks'].shape[-1]):
            pred_masks_resized[:, :, i] = cv2.resize(
                r['masks'][:, :, i].astype(np.uint8),
                (gt_mask.shape[1], gt_mask.shape[0]),
                interpolation=cv2.INTER_NEAREST
            )
        pred_masks = pred_masks_resized
    else:
        pred_masks = r['masks']

    # Passer les images sans données valides
    if gt_class_ids.size == 0 or r['class_ids'].size == 0:
```

```

        continue

    # Calculer l'AP pour l'image actuelle
    AP, precisions, recalls, overlaps = compute_ap(
        gt_bbox,          # Boîtes englobantes au sol
        gt_class_ids,     # IDs des classes au sol
        gt_mask,          # Masques au sol
        r['rois'],         # Boîtes englobantes prédites
        r['class_ids'],    # IDs des classes prédites
        r['scores'],       # Scores de confiance prédits
        pred_masks         # Masques prédits
    )

    APs.append(AP)

# Calcul et affichage de la moyenne de la précision moyenne (mAP)
if len(APs) > 0:
    mAP = np.mean(APs)
    print(f"Mean Average Precision (mAP): {mAP:.4f}")
else:
    print("Aucune prédiction valide trouvée pour l'évaluation.")

```

- **Résumé :** L'objectif principal de ce script est de quantifier la performance d'un modèle d'instance segmentation en calculant la précision moyenne des prédictions à travers toutes les images test. Cela permet de mesurer l'efficacité du modèle à prédire correctement les objets dans des images complexes.
- **Commentaire :** Le modèle a obtenu un score de mAP de 0.9333, ce qui signifie qu'il a une performance très élevée pour la détection et la segmentation des objets. Ce résultat indique que le modèle réussit à localiser et segmenter les objets avec une précision de 93,33%, montrant une bonne capacité à généraliser sur les images de test. Un tel score suggère que le modèle est fiable et efficace pour des tâches complexes de segmentation d'instances.

#### g) Évaluation des Performances du Modèle de Segmentation d'Instances avec mAP, Précision, Rappel et F1 Score :

```

from mrcnn.utils import compute_ap, extract_bboxes
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score
import cv2
import json
from datetime import datetime

# Liste pour stocker les métriques pour chaque image
APs = []
all_true_labels = []
all_pred_labels = []

```

```
# Parcourir toutes les images du dataset de test
for image_id in dataset_test.image_ids:
    # Charger l'image
    image = dataset_test.load_image(image_id)

    # Charger les données au sol (ground truth): masques et IDs de
    classe
    gt_mask, gt_class_ids = dataset_test.load_mask(image_id)

    # Générer les boîtes englobantes au sol
    gt_bbox = extract_bboxes(gt_mask)

    # Effectuer l'inférence sur l'image
    results = inference_model.detect([image], verbose=0)
    r = results[0]

    # Passer les images sans prédictions
    if r['masks'].size == 0:
        continue

    # Redimensionner les masques prédits si nécessaire
    if r['masks'].shape[:2] != gt_mask.shape[:2]:
        pred_masks_resized = np.zeros(gt_mask.shape, dtype=np.uint8)
        for i in range(r['masks'].shape[-1]):
            pred_masks_resized[:, :, i] = cv2.resize(
                r['masks'][:, :, i].astype(np.uint8),
                (gt_mask.shape[1], gt_mask.shape[0]),
                interpolation=cv2.INTER_NEAREST
            )
        pred_masks = pred_masks_resized
    else:
        pred_masks = r['masks']

    # Passer les images sans données au sol ou sans prédictions valides
    if gt_class_ids.size == 0 or r['class_ids'].size == 0:
        continue

    # Calculer l'AP pour l'image actuelle
    AP, precisions, recalls, overlaps = compute_ap(
        gt_bbox,          # Boîtes englobantes au sol
        gt_class_ids,     # IDs des classes au sol
        gt_mask,          # Masques au sol
        r['rois'],        # Boîtes englobantes prédites
        r['class_ids'],   # IDs des classes prédites
        r['scores'],      # Scores de confiance prédits
        pred_masks        # Masques prédits
    )
```

```
APs.append(AP)

# Préparer les étiquettes pour sklearn
# Convertir les masques en labels binaires par pixel
gt_mask_binary = np.any(gt_mask > 0, axis=2).astype(np.int32)
pred_mask_binary = np.any(pred_masks > 0, axis=2).astype(np.int32)

# Aplatir les masques pour les métriques sklearn
gt_mask_flat = gt_mask_binary.flatten()
pred_mask_flat = pred_mask_binary.flatten()

# Stocker les étiquettes aplaties
all_true_labels.extend(gt_mask_flat)
all_pred_labels.extend(pred_mask_flat)

# Convertir les listes en tableaux numpy pour sklearn
all_true_labels = np.array(all_true_labels)
all_pred_labels = np.array(all_pred_labels)

# Calcul des métriques globales
if len(APs) > 0:
    # Calculer mAP
    mAP = np.mean(APs)
    print(f"Mean Average Precision (mAP): {mAP:.4f}")

    # Calculer les métriques sklearn
    precision = precision_score(all_true_labels, all_pred_labels,
average='binary')
    recall = recall_score(all_true_labels, all_pred_labels,
average='binary')
    f1 = f1_score(all_true_labels, all_pred_labels, average='binary')

    # Afficher toutes les métriques
    print("\nOverall Metrics:")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Statistiques des AP par image
    print("\nPer-image AP Statistics:")
    print(f"Min AP: {np.min(APs):.4f}")
    print(f"Max AP: {np.max(APs):.4f}")
    print(f"Median AP: {np.median(APs):.4f}")
    print(f"Standard Deviation AP: {np.std(APs):.4f}")

    # Distribution détaillée des APs
```

```
percentiles = [25, 50, 75, 90, 95]
print("\nAP Percentiles:")
for p in percentiles:
    print(f"{p}th percentile: {np.percentile(APs, p):.4f}")
else:
    print("No valid predictions found for evaluation")

# Sauvegarder les métriques dans un fichier JSON
metrics = {
    "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    "mAP": float(mAP) if len(APs) > 0 else None,
    "precision": float(precision) if len(APs) > 0 else None,
    "recall": float(recall) if len(APs) > 0 else None,
    "f1_score": float(f1) if len(APs) > 0 else None,
    "num_images_evaluated": len(APs),
    "ap_statistics": {
        "min": float(np.min(APs)) if len(APs) > 0 else None,
        "max": float(np.max(APs)) if len(APs) > 0 else None,
        "median": float(np.median(APs)) if len(APs) > 0 else None,
        "std": float(np.std(APs)) if len(APs) > 0 else None,
        "percentiles": {
            str(p): float(np.percentile(APs, p)) for p in percentiles
        }
    }
}

if len(APs) > 0:
    pass

with open('evaluation_metrics.json', 'w') as f:
    json.dump(metrics, f, indent=4)
print("\nMetrics saved to 'evaluation_metrics.json'")
```



- **Résumé du Code :** Le code évalue la performance du modèle de segmentation d'instances en calculant des métriques telles que le mAP (Mean Average Precision), la précision, le rappel et le F1 score. Il analyse les prédictions du modèle pour chaque image du dataset de test, calcule l'AP pour chaque image, puis extrait des statistiques globales sur les performances du modèle. Les résultats sont sauvegardés dans un fichier JSON pour un suivi détaillé.
- **Commentaire des Résultats :** Le mAP de 0.9333 indique une performance excellente du modèle, avec une précision de 91,80%, un rappel de 89,74% et un F1 score de 90,76%. Ces scores montrent que le modèle est performant dans la détection et la segmentation des objets. Les statistiques des APs montrent une grande majorité d'images avec des résultats parfaits (AP = 1), bien que quelques images aient des APs plus faibles. Les percentiles montrent que la majorité des images ont une performance optimale, avec très peu d'erreurs. En résumé, le modèle est globalement très performant pour la tâche de segmentation d'instances.

## Conclusion :

En conclusion, l'utilisation de Mask R-CNN pour la segmentation du koala a montré son efficacité en permettant d'extraire avec précision les contours et les masques associés à cet animal. Les résultats obtenus confirment la robustesse de cette méthode, même dans des environnements complexes, tout en soulignant l'importance d'un ensemble de données diversifié pour améliorer la généralisation du modèle.