



---

**UFR des Sciences et Technologies**  
**Master 2 des signaux et image en médecine**

**Rapport du Projet :**

**Segmentation d'Images de Chevaux : Développement d'un Modèle de Segmentation**  
**Basé sur Mask R-CNN**

**Réalisé par Imane EL BLAGÉ**

**Encadré : Delphine Maugars**

## Introduction :

La segmentation d'images est une tâche essentielle en vision par ordinateur, permettant de comprendre en détail la structure d'une image en assignant une classe spécifique à chaque pixel. Ce projet met en œuvre un pipeline complet pour :

1. Charger des données d'images et de masques associés.
2. Évaluer les performances d'un modèle préentraîné Mask R-CNN.
3. Visualiser les résultats pour une meilleure interprétation des prédictions.

L'objectif est de fournir une méthodologie répliquable pour d'autres tâches de segmentation d'images.

## Objectifs :

### Objectifs généraux :

- Tester un modèle Mask R-CNN préentraîné sur un dataset d'images annotées.
- Analyser les performances du modèle en termes de segmentation.
- Mettre en place un pipeline automatisé, incluant le chargement des données, la prédiction et l'évaluation.

### Objectifs spécifiques :

- Mettre en œuvre des métriques d'évaluation (IoU, Dice Score) pour quantifier la qualité des prédictions.
- Produire des visualisations claires pour illustrer les résultats du modèle.
- Garantir un traitement uniforme des images en termes de dimensions et de normalisation.

## Données :

- **Source des données :** DATASET COCO 2017.
- **Annotations utilisées :** Fichiers d'annotations JSON pour les images d'entraînement et de validation.
- **Prétraitement :**
  - Chargement des images depuis les chemins fournis.
  - Extraction des annotations spécifiques à la catégorie "cheval".
  - Vérification de l'intégrité des données (existence des fichiers, format compatible).

## Méthodologie :

- **Outils et bibliothèques :**

- *Python 3.11*
- *Open CV* pour le traitement des images.
- *TensorFlow/Keras* pour la construction et l'entraînement du modèle.
- *pycocotools* pour manipuler les annotations COCO.

```

Invite de commandes - p x + v
Downloading cffi-1.17.1-cp311-cp311-win_amd64.whl (181 kB)
Downloading jsonpointer-3.0.0-py2.py3-none-any.whl (7.6 kB)
Downloading webcolors-24.11.1-py3-none-any.whl (14 kB)
Downloading fqdn-1.5.1-py3-none-any.whl (9.1 kB)
Downloading isoduration-20.11.0-py3-none-any.whl (11 kB)
Downloading uri-template-1.3.0-py3-none-any.whl (11 kB)
Downloading arrow-1.3.0-py3-none-any.whl (66 kB)
Downloading pyparser-2.22-py3-none-any.whl (117 kB)
Downloading types_python_dateutil-2.9.0.20241003-py3-none-any.whl (9.7 kB)
Building wheels for collected packages: tensorflow-object-detection-api
  Building wheel for tensorflow-object-detection-api (setup.py) ... done
  Created wheel for tensorflow-object-detection-api: filename=tensorflow_object_detection_api-2.11.1-py3-none-any.whl size=844494 sha256=14ca7f6045cd9b6bf4ce4836cb042c176972c8e5a65d779b6f80679
  Stored in directory: c:\users\oassa\appdata\local\pip\cache\wheels\cf\4f\46\c7a165b6d85971257a7636363636363636363636363636363636363636363636
Successfully built tensorflow-object-detection-api
Installing collected packages: webencodings, wcwidth, pywin32, pure-eval, nh3, fastjsonschema, zip, widgetencodings,
websocket-client, webcolors, uri-template, types-python-dateutil, traitlets, tornado, tynccore, supabase, sqlalchemy, san
dtrush, xpdfs-py, rf3986-validator, rf3986, rf3339-validator, pyms, pyyaml, pygments, pywin32-crypto, python-jan-
gger, python-dateutil, pyparsing, pyparser, psutil, prompt-toolkit, prometheus-client, pschmains, piparim, pldown, p
arso, pandocfilters, overrides, nest-asyncio, more-itertools, mistune, lxml, lxmlhtml, jupyterlab-widgets, jupyterlab
pygments, jsonpointer, json5, Jinja2, h11, fqdn, fonttools, executing, docutils, defusedxml, decouple, debugpy, Cython,
cycler, contourpy, contextlib2, colorama, bleach, bcbioports, turfile, babel, attrs, async-lru, aiohttp, tomlkit, de
ck-data, requests-toolbelt, referencing, readme-renderer, matplotlib-inline, matplotlib, jupyter-core, jlab, jupyter-lab-
tools, jupyter-context, jupyterclasses, Importlib-metadata, httpcore, com, cffi, beautifulsoup, arrow, anyio, httpx,
jupyter-server-terminals, jupyter-client, jsonschema-specifications, jsonschema, python-httpx, agent-cffi-binding,
twine, jsonschema, ipynbwidgets, ipynbkernel, argon2-cffi, shiboken, jupyter-console, nbclient, jupyter-events, document, j
upyter-server, notebook-shim, jupyterlab-server, jupyter-lsp, jupyterlab, notebook, jupyter, tensorflow-object-detection
-api

```

## **CODE 1 :**

- Visualisation des Masques avec PYCOCOTOOLS.

### **1. Objectif du Code :**

Le premier code a pour but de :

- Identifier les images contenant des chevaux dans un ensemble d'annotations au format COCO.
- Charger ces images depuis leur chemin d'accès.
- Superposer les masques de segmentation des chevaux sur ces images pour une visualisation simple et intuitive.

### **2. Fonctionnalités du Code :**

#### **Chargement des annotations COCO**

Le fichier `instances_train2017.json` contient des annotations au format COCO, qui incluent :

- Les catégories disponibles (par exemple, cheval, voiture, personne ...)
- Les informations sur les images telles que leurs noms de fichier et dimensions
- Les masques de segmentation pour chaque objet annoté

#### **Filtrage pour les chevaux**

Le code :

- Identifie la catégorie *"horse"* en recherchant dans les annotations.
- Récupère les identifiants des images contenant des chevaux.

#### **Chargement d'une image et de ses masques**

Une fois une image sélectionnée :

- Le code charge cette image.
- Les masques associés aux chevaux dans cette image sont également récupérés.

#### **Visualisation des résultats**

Le code colore les zones correspondant aux chevaux en rouge, permettant de vérifier visuellement les annotations sur l'image.

### 3. Structure du Code :

#### ➤ Explications du Code :

##### 1. Importation des bibliothèques

```
from pycocotools.coco import COCO
import matplotlib.pyplot as plt
import cv2
import numpy as np
from mrcnn.utils import Dataset
from PIL import Image
import os
```

Ces bibliothèques servent à :

- Gérer les annotations COCO (*pycocotools*).
- Traiter les images (*OpenCV*, *Pillow*).
- Afficher les résultats (*matplotlib*).

##### 2. Chargement des annotations COCO

```
# PATH TO fichier_instances
fichier_instances = 'C:/Users/oassa/Downloads/PROJET IMANE/annotations/instances_train2017.json'

# let's load instance annotations
coco = COCO(fichier_instances)
```

- ☐ Le fichier JSON contient des annotations détaillées (catégories, masques..)
- ☐ L'objet COCO permet d'interagir avec ces données.

##### 3. Extraction des catégories et des images :

```
# There are many categories, we are searching for HORSES
categories = coco.loadCats(coco.getCatIds())
horse_names = [horse['name'] for horse in categories]
print("Catégories disponibles :", horse_names)

# As we have many IDs, let's extract only 'horse' IDs
horse_ids = coco.getCatIds(catNms=['horse'])
image_ids = coco.getImgIds(catIds=horse_ids)

# loading the images
images = coco.loadImgs(image_ids)
print(f"Nombre d'images contenant des chevaux : {len(images)}")
```

- loadCats charge toutes les catégories disponibles.
- getCatIds filtre pour n'obtenir que les chevaux.
- getImgIds récupère les identifiants des images contenant des chevaux.

#### 4. Chargement et affichage d'une image :

```
if len(image_ids) > 0:
    image_id = image_ids[0] # Utiliser la première image contenant des chevaux
    image_info = coco.loadImgs([image_id])[0]
    image_path = f'C:/Users/oassa/Downloads/PROJET IMANE/train2017/train2017/{image_info["file_name"]}'

    # Vérification si l'image existe
    try:
        image = cv2.imread(image_path)
```

- loadImgs récupère les informations sur une image (chemin, dimensions).
- cv2.imread charge l'image depuis le disque.

#### 5. Superposition des masques :

```
# Charger les annotations pour cette image
annotation_ids = coco.getAnnIds(imgIds=[image_id], catIds=horse_ids)
annotations = coco.loadAnns(annotation_ids)

# Dessiner les masques sur l'image
for ann in annotations:
    mask = coco.annToMask(ann)
    image[mask == 1] = [255, 0, 0] # Colorer les masques en rouge
```

- getAnnIds et loadAnns récupèrent les annotations spécifiques à l'image.
- annToMask convertit chaque annotation en masque binaire (pixels 0/1).
- Les pixels correspondant aux masques sont colorés en rouge.

#### 6. Affichage des résultats

```
# Afficher l'image avec les annotations
plt.imshow(image)
plt.axis('off')
plt.show()
```

- La librairie matplotlib affiche l'image annotée.

#### 4. Résultats Obtenus :

```
PS C:\Users\oassa> & C:/Users/oassa/AppData/Local/Programs/Python/Python311/python.exe  
"c:/Users/oassa/Downloads/PROJET IMANE/Cheval_part1 Classification.py"
```

```
2024-11-18 01:04:12.729667: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You  
may see slightly different numerical results due to floating-point round-off errors from different  
computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.  
2024-11-18 01:04:16.201076: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You  
may see slightly different numerical results due to floating-point round-off errors from different  
computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.  
loading annotations into memory...
```

```
Done (t=18.59s)
```

```
creating index...
```

```
index created!
```

```
Catégories disponibles : ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat',  
'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',  
'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis',  
'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',  
'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli',  
'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv',  
'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator',  
'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']
```

```
Nombre d'images contenant des chevaux : 2941
```



### ➤ Chargement des annotations :

**Le fichier COCO a été chargé avec succès, et toutes les 80 catégories disponibles ont été identifiées.**

### ➤ Images contenant des chevaux :

- **2941 images ont été détectées avec des annotations associées à la catégorie "cheval".**
- **Les annotations sont exploitables pour des tâches comme la segmentation ou la détection.**

### ➤ Visualisation des masques :

**Une image a été sélectionnée et les masques correspondant aux chevaux ont été correctement affichés en superposition rouge.**

### **En résumé :**

Les résultats sont globalement satisfaisants, Le code remplit son objectif de charger, extraire et visualiser les annotations COCO pour la catégorie "cheval" Quelques optimisations pourraient rendre le processus plus robuste et dynamique.



## CODE 02 :

### ➤ Objectifs :

Ce code met en œuvre un modèle de segmentation basé sur la méthode Mask R-CNN visant à détecter et segmenter des objets d'intérêt (ici des chevaux) dans des images. Les principales étapes incluent :

1. Chargement et prétraitement des images et des annotations.
2. Création d'un modèle Mask R-CNN basé sur une architecture ResNet50.
3. Entraînement supervisé sur un dataset d'entraînement avec validation.
4. Sauvegarde du modèle et des résultats pour utilisation future.

### ➤ Code :

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.applications import ResNet50
from pycocotools.coco import COCO

print(f"TensorFlow version: {tf.__version__}")

class horseConfig:
    """Configuration pour le modèle de segmentation d'chevaux"""
    NAME = "horse_segmentation"
    IMAGES_PER_GPU = 2
    BATCH_SIZE = IMAGES_PER_GPU
    STEPS_PER_EPOCH = 100
    VALIDATION_STEPS = 50
    LEARNING_RATE = 0.001
    NUM_CLASSES = 2 # Background + horse
    IMAGE_SIZE = (640, 480)
    BACKBONE = 'resnet50'

    # Chemins des données
    TRAIN_ANNOTATIONS = 'C:/Users/oassa/Downloads/PROJET IMANE/annotations/instances_train2017.json'
    TRAIN_IMAGES = 'C:/Users/oassa/Downloads/PROJET IMANE/train2017/train2017'
    VAL_ANNOTATIONS = 'C:/Users/oassa/Downloads/PROJET IMANE/annotations/instances_val2017.json'
    VAL_IMAGES = 'C:/Users/oassa/Downloads/PROJET IMANE/val2017'
```

```

class ResizeLayer(layers.Layer):
    """Couche personnalisée pour le redimensionnement des images"""
    def __init__(self, target_size, **kwargs):
        super(ResizeLayer, self).__init__(**kwargs)
        self.target_size = target_size

    def call(self, inputs):
        return tf.image.resize(inputs, self.target_size)

    def get_config(self):
        config = super(ResizeLayer, self).get_config()
        config.update({"target_size": self.target_size})
        return config

class horseDataset:
    """Classe pour gérer le dataset d'chevaux"""
    def __init__(self, annotation_path, images_dir):
        """Initialise le dataset"""
        self.images_dir = images_dir
        try:
            self.coco = COCO(annotation_path)
            print(f"Chargement réussi des annotations depuis {annotation_path}")
        except Exception as e:
            print(f"Erreur lors du chargement des annotations: {e}")
            raise

        self.horse_ids = self.coco.getCatIds(catNms=['horse'])
        if not self.horse_ids:
            print("Attention: Aucune catégorie 'horse' trouvée dans les annotations")
            print("Catégories disponibles:", self.coco.loadCats(self.coco.getCatIds()))

```

```

        self.img_ids = self.coco.getImgIds(catIds=self.horse_ids)
        print(f"Nombre d'images trouvées: {len(self.img_ids)}")

    def load_data(self, max_images=None):
        """Charge et prépare les données"""
        images = []
        masks = []
        valid_image_count = 0

        img_ids = self.img_ids[:max_images] if max_images else self.img_ids

        for img_id in img_ids:
            try:
                # Charger l'image
                img_info = self.coco.loadImgs([img_id])[0]
                image_path = os.path.join(self.images_dir, img_info["file_name"])

                if not os.path.exists(image_path):
                    print(f"Image non trouvée: {image_path}")
                    continue

                # Charger et prétraiter l'image
                image = tf.io.read_file(image_path)
                image = tf.image.decode_jpeg(image, channels=3)
                image = tf.cast(image, tf.float32)
                image = tf.image.resize(image, horseConfig.IMAGE_SIZE)

                # Charger les annotations

```

```

# Charger les annotations
anns_ids = self.coco.getAnnIds(imgIds=img_id, catIds=self.horse_ids, iscrowd=False)
anns = self.coco.loadAnns(anns_ids)

if not anns:
    continue

# Créer un masque binaire
mask = np.zeros(horseConfig.IMAGE_SIZE + (2,), dtype=np.float32)

for ann in anns:
    # Convertir le masque COCO en masque binaire
    m = self.coco.annToMask(ann)
    m = tf.image.resize(
        m[...], np.newaxis],
        horseConfig.IMAGE_SIZE,
        method='nearest'
    ).numpy()
    # Mettre à jour le canal du cheval (classe 1)
    mask[..., 1] = np.maximum(mask[..., 1], m[..., 0])

# Le fond (classe 0) est l'inverse du masque du cheval
mask[..., 0] = 1 - mask[..., 1]

images.append(image.numpy())
masks.append(mask)
valid_image_count += 1

```

```

if valid_image_count % 100 == 0:
    print(f"Chargées {valid_image_count} images valides")

except Exception as e:
    print(f"Erreur lors du traitement de l'image {img_id}: {e}")
    continue

if not images:
    raise ValueError("Aucune image valide n'a été chargée!")

print(f"Chargement terminé: {len(images)} images valides traitées")
return np.array(images), np.array(masks)

def create_mask_rcnn_model(config):
    """Crée un modèle de segmentation basé sur ResNet50"""
    # Backbone ResNet50
    backbone = ResNet50(
        include_top=False,
        weights='imagenet',
        input_shape=config.IMAGE_SIZE + (3,)
    )

    # Feature Pyramid Network
    C5 = backbone.get_layer('conv5_block3_out').output
    P5 = layers.Conv2D(256, (1, 1), name='fpn_c5p5')(C5)
    P5_upsampled = layers.UpSampling2D(size=(2, 2), name='fpn_p5upsampled')(P5)

```

```

C4 = backbone.get_layer('conv4_block6_out').output
P4 = layers.Conv2D(256, (1, 1), name='fpn_c4p4')(C4)
P4 = layers.Add(name='fpn_p4add')([P5_upsampled, P4])

# Mask head
x = layers.Conv2D(256, (3, 3), padding="same", activation="relu")(P4)
x = layers.Conv2D(256, (3, 3), padding="same", activation="relu")(x)
x = layers.Conv2D(256, (3, 3), padding="same", activation="relu")(x)
x = layers.Conv2D(256, (3, 3), padding="same", activation="relu")(x)

# Upsampling progressif
x = layers.UpSampling2D(size=(2, 2))(x)
x = layers.Conv2D(128, (3, 3), padding="same", activation="relu")(x)

x = layers.UpSampling2D(size=(2, 2))(x)
x = layers.Conv2D(64, (3, 3), padding="same", activation="relu")(x)

x = layers.UpSampling2D(size=(2, 2))(x)
x = layers.Conv2D(32, (3, 3), padding="same", activation="relu")(x)

x = layers.UpSampling2D(size=(2, 2))(x)
x = layers.Conv2D(16, (3, 3), padding="same", activation="relu")(x)

# Utiliser ResizeLayer pour le redimensionnement final
x = ResizeLayer(config.IMAGE_SIZE)(x)

# Couche de sortie
mask_output = layers.Conv2D(
    config.NUM_CLASSES, (1, 1),

```

```

# Couche de sortie
mask_output = layers.Conv2D(
    config.NUM_CLASSES, (1, 1),
    activation='softmax',
    name='mask_output'
)(x)

# Créer et compiler le modèle
model = Model(inputs=backbone.input, outputs=mask_output)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=config.LEARNING_RATE),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

print(f"Input shape: {config.IMAGE_SIZE + (3,)}")
print(f"Output shape: {model.output_shape}")

return model

def create_train_callbacks(model_name):
    """Crée les callbacks pour l'entraînement"""
    return [
        tf.keras.callbacks.ModelCheckpoint(
            f'{model_name}_best.keras',
            save_best_only=True,
            monitor='val_loss'
        ),

```

```

def create_train_callbacks(model_name):
    ),
    tf.keras.callbacks.EarlyStopping(
        patience=3,
        monitor='val_loss',
        restore_best_weights=True
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.1,
        patience=2,
        min_lr=1e-6
    ),
    tf.keras.callbacks.TensorBoard(
        log_dir=f'./logs/{model_name}',
        update_freq='epoch'
    )
]

def main():
    """Fonction principale d'entraînement"""
    # Configuration
    config = horseConfig()

    try:
        # Charger et préparer les données d'entraînement
        print("Chargement du dataset d'entraînement...")
        train_dataset = horseDataset(
            annotation_path=config.TRAIN_ANNOTATIONS,
            images_dir=config.TRAIN_IMAGES

```

```

X_train, y_train = train_dataset.load_data(max_images=1000)

# Charger et préparer les données de validation
print("Chargement du dataset de validation...")
val_dataset = horseDataset(
    annotation_path=config.VAL_ANNOTATIONS,
    images_dir=config.VAL_IMAGES
)
X_val, y_val = val_dataset.load_data(max_images=200)

# Normaliser les images
print("Normalisation des données...")
X_train = X_train / 255.0
X_val = X_val / 255.0

# Créer le modèle
print("Création du modèle...")
model = create_mask_rcnn_model(config)

# Créer les callbacks
callbacks = create_train_callbacks(config.NAME)

# Entraîner le modèle
print("Début de l'entraînement...")
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20,
    batch_size=config.BATCH_SIZE,

```

```

# Créer les callbacks
callbacks = create_train_callbacks(config.NAME)

# Entraîner le modèle
print("Début de l'entraînement...")
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20,
    batch_size=config.BATCH_SIZE,
    callbacks=callbacks,
    verbose=1
)

# Sauvegarder le modèle final
model.save(f'{config.NAME}_final.keras')
print("Entraînement terminé et modèle sauvegardé!")

# Sauvegarder l'historique d'entraînement
np.save(f'{config.NAME}_history.npy', history.history)

except Exception as e:
    print(f"Erreur dans le processus: {e}")
    raise

if __name__ == "__main__":
    main()

```

## Explication du Code :

### 1. Configuration du Modèle (horseConfig)

La classe horseConfig configure les paramètres globaux nécessaires pour entraîner le modèle :

- Taille des images : (640, 480).
- Nombre de classes : 2 (cheval et arrière-plan).
- Chemins des données : Spécifie les fichiers d'annotations et les répertoires d'images pour les ensembles d'entraînement et de validation.

### 2. Gestion des Données (horseDataset)

La classe horseDataset gère le chargement et la préparation des données :

- Initialisation : Charge les annotations au format COCO et extrait uniquement celles liées à la catégorie "cheval".
- Méthode load\_data :
  - Charge les images et leurs masques correspondants.
  - Prétraitement :
    - Redimensionne les images à la taille spécifiée dans la configuration.
    - Génère un masque binaire :
      - Canal 1 : masque du cheval.
      - Canal 0 : arrière-plan (inverse du masque du cheval).

### 3. Redimensionnement des Images (ResizeLayer)

Une couche personnalisée pour redimensionner les images à la taille cible. Elle utilise (tf.image.resize) pour garantir que toutes les sorties soient de dimensions cohérentes.

### 4. Création du Modèle (create\_mask\_rcnn\_model)

Cette fonction crée un modèle de segmentation basé sur ResNet50 et une pyramide de caractéristiques (FPN) :

- Backbone ResNet50 :
  - Prend les images en entrée et extrait des caractéristiques.
  - Supprime les couches de classification initiales.
- Feature Pyramid Network (FPN) :
  - Combine les caractéristiques des différentes couches du ResNet50 pour capturer des détails à plusieurs échelles.
- Tête de masque (Mask Head) :
  - Effectue plusieurs convolutions pour affiner les masques.
  - Effectue un redimensionnement progressif (par UpSampling) pour revenir à la taille de l'image d'origine.
- Sortie finale :
  - Une carte binaire avec deux canaux (cheval et arrière-plan).
- Compilation :
  - Fonction de perte : categorical\_crossentropy.
  - Optimiseur : Adam avec un taux d'apprentissage configurable.

### 5.Callbacks (create\_train\_callbacks)

La fonction crée des mécanismes pour gérer l'entraînement :

- ModelCheckpoint : Sauvegarde le meilleur modèle.
- EarlyStopping : Arrête l'entraînement si la validation ne s'améliore plus.
- ReduceLROnPlateau : Diminue le taux d'apprentissage en cas de stagnation.
- TensorBoard : Permet de visualiser les métriques d'entraînement.

### 6.Fonction Principale (main)

La fonction main coordonne tout le processus :

- Chargement des données :
  - Charge et prépare les données d'entraînement et de validation en utilisant horseDataset.
  - Normalise les images (divise les valeurs des pixels par 255).
- Création du modèle :
  - Construit le modèle à partir des spécifications définies dans horseConfig.
- Entraînement :

- Entraîne le modèle avec les données chargées, les callbacks définis, et une configuration de 20 époques.
- Sauvegarde :
  - Sauvegarde le modèle final et l'historique d'entraînement.

## Résultats :

```

C:\Users\oassa> Downloads\PROJET IMANE> Cheval_part1 Classification.py ...
1 #PROJET COCO PROJECT
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Chargées 200 images valides
Chargées 300 images valides
Chargées 400 images valides
Chargées 500 images valides
Chargées 600 images valides
Chargées 700 images valides
Chargées 800 images valides
Chargées 900 images valides
Chargées 1000 images valides
Chargement terminé: 1000 images valides traitées
Chargement du dataset de validation...
loading annotations into memory...
Done (t=1.48s)
creating index...
Index created
Chargement réussi des annotations depuis C:/Users/oassa/Downloads/PROJET IMANE/annotations/instances_val2017.json
Nombre d'images trouvées: 128
Chargées 100 images valides
Chargement terminé: 128 images valides traitées
Normalisation des données...
WARNING:tensorflow:from C:\Users\oassa\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend\tensorflow\core.py:204: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

Input shape: (640, 480, 3)
Output shape: (None, 640, 480, 2)
Début de l'entraînement...
Epoch 1/20
500/500 2663s 5s/step - accuracy: 0.8646 - loss: 0.4982 - val_accuracy: 0.8868 - val_loss: 0.3556 - learning_rate: 0.0010
Epoch 2/20
500/500 2883s 6s/step - accuracy: 0.8664 - loss: 0.3200 - val_accuracy: 0.8534 - val_loss: 0.3485 - learning_rate: 0.0010
Epoch 3/20
500/500 2876s 6s/step - accuracy: 0.8687 - loss: 0.3039 - val_accuracy: 0.8980 - val_loss: 0.2269 - learning_rate: 0.0010
Epoch 4/20
500/500 2848s 6s/step - accuracy: 0.8733 - loss: 0.2992 - val_accuracy: 0.8902 - val_loss: 0.2638 - learning_rate: 0.0010
Epoch 5/20
147/500 4:11:56 43s/step - accuracy: 0.8732 - loss: 0.2994
  
```

L'entraînement du modèle Mask R-CNN sur un ensemble de données contenant 1000 images pour l'entraînement et 128 images pour la validation a présenté des difficultés techniques, notamment liées au stockage ou aux limitations matérielles. Malgré ces défis, le processus d'entraînement a pu progresser sur 5 époques, mais a été très long (plus de **4 heures**) et semble avoir stagné à certains moments.

## Analyse de l'exécution :

### 1. Chargement des données :

- Le code a chargé avec succès **1000 images** pour l'entraînement et **128 images** pour la validation.
- Les annotations COCO ont été traitées correctement et les données ont été normalisées pour s'adapter à l'entrée du modèle (forme (640, 480, 3))

### 2. Progrès de l'entraînement :

- **Durée par époque :**
  - Chaque époque a pris environ **44 à 48 minutes**.
  - Cela suggère que le processus est limité par les performances matérielles
- **Évolution des métriques :**
  - Précision d'entraînement (accuracy) : Stagne légèrement autour de **86-87%**.
  - Précision de validation (val\_accuracy) : Les fluctuations sont modérées, avec une meilleure valeur à **89.8%** lors de la troisième époque.



- La perte diminue initialement, mais atteint un plateau dès la troisième époque :
  - Perte d'entraînement : **0.4982** → **0.2994**.
  - Perte de validation : **0.3556** → **0.2994**.

### 3. Problèmes rencontrés :

- **Temps d'exécution très long :**
  - Plus de 4 heures pour seulement 5 époques est un indicateur que l'environnement d'exécution n'est pas optimisé pour ce type de modèle.
- **Stagnation des performances :**
  - Après la troisième époque, il semble y avoir peu d'amélioration dans les métriques, ce qui pourrait indiquer un besoin d'ajustement des hyperparamètres ou d'amélioration des données.

### 4. Avertissement TensorFlow :

Le message concernant **tf.placeholder** indique que vous utilisez des composants TensorFlow dépréciés. Cela n'affecte pas directement l'entraînement, mais il serait utile de mettre à jour le code pour éviter d'éventuels problèmes de compatibilité future :

- Remplacer `tf.placeholder` par son équivalent compatible avec TensorFlow v2 : `tf.compat.v1.placeholder`.

## CODE 03 :

Analyse détaillée et explication du code :

Code :

```

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
import os

class MaskRCNNConfig:
    IMAGE_SIZE = (640, 480)
    VAL_ANNOTATIONS = 'C:/Users/oassa/Downloads/PROJET IMANE/annotations/instances_val2017.json'
    VAL_IMAGES = 'C:/Users/oassa/Downloads/PROJET IMANE/val2017'

class ResizeLayer(tf.keras.layers.Layer):
    def __init__(self, target_size=(640, 480), **kwargs):
        super(ResizeLayer, self).__init__(**kwargs)
        self.target_size = target_size

    def call(self, inputs):
        return tf.image.resize(inputs, self.target_size)

    def get_config(self):
        config = super(ResizeLayer, self).get_config()
        config.update({
            'target_size': self.target_size
        })
        return config

    @classmethod
    def from_config(cls, config):
        return cls(**config)

```

```

# Le reste du code reste inchangé
class MaskRCNNDataset:
    def __init__(self, annotation_path, images_dir):
        self.annotation_path = annotation_path
        self.images_dir = images_dir

    def load_data(self, max_images=None):
        """Charge les données d'images et de masques"""
        image_paths = tf.io.gfile.glob(f"{self.images_dir}/*.jpg")
        if max_images:
            image_paths = image_paths[:max_images]

        images = []
        masks = []

        for image_path in image_paths:
            image = load_and_preprocess_image(image_path)
            if image is not None:
                images.append(image)
                mask_path = image_path.replace('images', 'annotations').replace('.jpg', '_mask.png')
                mask = load_and_preprocess_mask(mask_path)
                if mask is not None:
                    masks.append(mask)

        return np.array(images), np.array(masks)

    def load_and_preprocess_image(image_path):
        try:

```

```

        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.cast(image, tf.float32)
        image = tf.image.resize(image, MaskRCNNConfig.IMAGE_SIZE)
        image = image / 255.0
        return image
    except Exception as e:
        print(f"Erreur lors du chargement de l'image {image_path}: {e}")
        return None

def load_and_preprocess_mask(mask_path):
    try:
        mask = tf.io.read_file(mask_path)
        mask = tf.image.decode_png(mask, channels=1)
        mask = tf.image.resize(mask, MaskRCNNConfig.IMAGE_SIZE, method='nearest')
        mask = tf.cast(mask, tf.float32)
        background = tf.cast(tf.equal(mask, 0), tf.float32)
        foreground = tf.cast(tf.greater(mask, 0), tf.float32)
        mask_onehot = tf.stack([background, foreground], axis=-1)
        return mask_onehot
    except Exception as e:
        print(f"Erreur lors du chargement du masque {mask_path}: {e}")
        return None

def visualize_prediction(image, mask, prediction, save_path=None, display=True):
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    axes[0].imshow(image)
    axes[0].set_title('Image Originale')

```

```

    axes[0].axis('off')

    axes[1].imshow(mask[..., 1], cmap='gray')
    axes[1].set_title('Masque Réel')
    axes[1].axis('off')

    axes[2].imshow(prediction[..., 1], cmap='gray')
    axes[2].set_title('Prédiction')
    axes[2].axis('off')

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path)
        plt.close()
    elif display:
        plt.show()

def evaluate_model(model_path='horse_segmentation_best.keras'):
    if not os.path.exists(model_path):
        print(f"Erreur : Le modèle {model_path} n'existe pas.")
        return

    custom_objects = {
        'ResizeLayer': ResizeLayer
    }

    try:
        model = load_model(model_path, custom_objects=custom_objects)

```

```

    print(f"Modèle chargé depuis {model_path}")
except Exception as e:
    print(f"Erreur lors du chargement du modèle : {e}")
    return

val_dataset = MaskRCNNDataset(
    annotation_path=MaskRCNNConfig.VAL_ANNOTATIONS,
    images_dir=MaskRCNNConfig.VAL_IMAGES
)
X_val, y_val = val_dataset.load_data(max_images=10)

predictions = model.predict(X_val)

iou_scores = []
dice_scores = []

for i in range(len(X_val)):
    true_mask = y_val[i, ..., 1]
    pred_mask = predictions[i, ..., 1] > 0.5

    intersection = np.logical_and(true_mask, pred_mask).sum()
    union = np.logical_or(true_mask, pred_mask).sum()
    iou = intersection / (union + 1e-7)

    dice = (2 * intersection) / (true_mask.sum() + pred_mask.sum() + 1e-7)

    iou_scores.append(iou)
    dice_scores.append(dice)

```

```

for i in range(len(X_val)):
    true_mask = y_val[i, ..., 1]
    pred_mask = predictions[i, ..., 1] > 0.5

    intersection = np.logical_and(true_mask, pred_mask).sum()
    union = np.logical_or(true_mask, pred_mask).sum()
    iou = intersection / (union + 1e-7)

    dice = (2 * intersection) / (true_mask.sum() + pred_mask.sum() + 1e-7)

    iou_scores.append(iou)
    dice_scores.append(dice)

    visualize_prediction(
        X_val[i],
        y_val[i],
        predictions[i],
        save_path=f'prediction_results_{i}.png',
        display=False
    )

print(f"IoU moyen: {np.mean(iou_scores):.4f}")
print(f"Dice Score moyen: {np.mean(dice_scores):.4f}")

if __name__ == "__main__":
    model_path = 'horse_segmentation_best.keras'
    evaluate_model(model_path)

```

Ce code implémente un pipeline de segmentation d'images utilisant TensorFlow et Keras. Voici une explication détaillée des différentes parties.

## 1. Importation des bibliothèques

Le code commence par importer des modules essentiels :

- tensorflow : utilisé pour construire, entraîner, et évaluer des modèles de deep learning.
- numpy : utilisé pour manipuler les données sous forme de tableaux.
- matplotlib.pyplot : pour visualiser les images, masques et prédictions.
- os : pour interagir avec le système de fichiers.

## 2. Classe MaskRCNNConfig

C'est une classe de configuration contenant des paramètres statiques :

- IMAGE\_SIZE : la taille cible des images après redimensionnement.
- VAL\_ANNOTATIONS et VAL\_IMAGES : chemins vers les annotations de validation et les images correspondantes.

**Rôle :** Facilite l'utilisation de constantes globales dans le reste du code.

## 3. Classe ResizeLayer

C'est une couche personnalisée TensorFlow pour redimensionner les images.

- **Méthodes :**
  - \_\_init\_\_ : Initialise la taille cible du redimensionnement.
  - call : Applique le redimensionnement via tf.image.resize.
  - get\_config et from\_config : Gèrent la sérialisation et désérialisation pour permettre de sauvegarder et recharger cette couche dans un modèle.

**Rôle :** Simplifie le prétraitement en intégrant le redimensionnement comme partie du modèle.

## 4. Classe MaskRCNNDataset

Cette classe gère le chargement des données d'images et de masques.

- **Attributs :**
  - annotation\_path : chemin vers le fichier d'annotations.
  - images\_dir : répertoire contenant les images.
- **Méthode load\_data(max\_images=None) :**
  - Charge un ensemble limité d'images (défini par max\_images).
  - Pour chaque image, un masque correspondant est recherché et chargé.
  - Retourne deux tableaux : un pour les images (images) et un pour les masques (masks).

## 5. Fonctions auxiliaires :

### 1-load\_and\_preprocess\_image(image\_path)

- Lit une image à partir de son chemin.
- Redimensionne l'image à MaskRCNNConfig.IMAGE\_SIZE.
- Normalise les pixels pour être dans l'intervalle [0, 1].

**Rôle :** Préparer les images en entrée pour le modèle.

### 2-load\_and\_preprocess\_mask(mask\_path)

- Lit un masque d'image binaire (format PNG) et le redimensionne.
- Crée une version **one-hot encodée** :
  - Canal 0 : arrière-plan.
  - Canal 1 : premier plan (zones avec des pixels non nuls).

**Rôle :** Préparer les masques pour être compatibles avec le modèle (deux classes : arrière-plan et objet).

### 3-visualize\_prediction

- Affiche ou enregistre une comparaison entre l'image originale, le masque réel, et le masque prédit.
- Utilise **matplotlib** pour générer une figure avec 3 sous-graphiques :
  - Image d'entrée.
  - Masque réel.
  - Masque prédit.

**Rôle :** Fournir des visualisations pour évaluer les performances du modèle.

## 6. Fonction principale evaluate\_model :

Cette fonction évalue un modèle Mask R-CNN sauvegardé, en le testant sur un ensemble de données de validation.

- **Chargement du modèle :**
  - Vérifie si le fichier du modèle existe.
  - Charge le modèle en incluant la couche personnalisée ResizeLayer.
- **Chargement des données de validation :**
  - Charge un sous-ensemble des images de validation et leurs masques via MaskRCNNDataset.
- **Prédiction :**
  - Effectue des prédictions sur les images de validation.

- Compare chaque prédiction avec le masque réel en calculant deux métriques :
  - **IoU (Intersection over Union)** : Mesure la superposition entre la vérité terrain et la prédiction.
  - **Dice Score** : Mesure l'équilibre entre précision et rappel.
- **Visualisation des résultats :**
  - Sauvegarde les comparaisons entre les masques réels et prédits sous forme d'images.
- **Rapport des métriques :**
  - Affiche les IoU et Dice Score moyens sur l'ensemble des images de validation.

## 7. Métriques utilisées :

- **IoU (Intersection over Union)** :  

$$\text{IoU} = \frac{\text{Intersection entre prédiction et vérité terrain}}{\text{Union entre prédiction et vérité terrain}}$$
- **Dice Score** :  

$$\text{Dice} = \frac{2 \times \text{Intersection}}{\text{Taille totale des deux ensembles} + \text{Intersection}}$$

## 8. Fonction principale main :

Le programme commence ici lorsqu'il est exécuté directement :

1. Spécifie le chemin du modèle (model\_path).
2. Appelle la fonction evaluate\_model pour charger et évaluer le modèle.



**UFR des Sciences et Technologies**  
**Master 2 des signaux et image en médecine**