**Rapport du Projet : Segmentation d’Images de Chevaux**

**Développement d’un Modèle de Segmentation Basé sur Mask R-CNN**

By imane EL BLAGE

Le 18/11/2024

**Introduction :**

* **Contexte :** Présentez l’importance de la segmentation d’images en vision par ordinateur avec un focus sur les applications pour détecter et segmenter des chevaux.
* **Objectif du projet :** Développer un modèle de segmentation capable d’identifier et de segmenter des chevaux dans des images à l’aide du framework COCO et d’un réseau ResNet50.

**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.

**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**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ces bibliothèques servent à :

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

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

   Description générée automatiquementChargement des annotations COCO**

 Le fichier JSON contient des annotations détaillées (catégories, masques..)

 L’objet COCO permet d’interagir avec ces données.

1. **Extraction des catégories et des images :**

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

Description générée automatiquement**

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

1. **Chargement et affichage d’une image :**

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

Description générée automatiquement**

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

1. **Superposition des masques :**

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

Description générée automatiquement**

* **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.**

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

   Description générée automatiquementAffichage des résultats**

* La librairie matplotlib affiche l'image annotée.

**4. Résultats Obtenus :**

* **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 :**

* 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.

1. **Résultats :**