Projet de Session

Département de génie logiciel et des TI

Projet de l'évaluation de la performance de la segmentation des méthodes SOLOv2, Mask R-CNN+ et U-Net pour la détection de déchets dans un environnement

Jacob Cossette

COSJ08079801

Professeur superviseur

Éric Granger

Date

19 décembre 2024

Suivi des changements

*__A__ – Ajouté **M** – Modifié **S** – Supprimé

NUMÉRO DE VERSION	DATE aaaa/mm/jj	NUMÉRO DE FIGURE, TABLE OU SECTION	A* M S	BRÈVE DESCRIPTION DU CHANGEMENT	NUMÉRO DE DEMANDE CHANGEMENT
1.0	2024/12/19		A	Publication Initial	

TABLE DES MATIÈRES

1. MISE EN SITUATION

1.1 Domaine d'application

La détection automatique des déchets pourrait être particulièrement utile pour classifier, analyser et trier les déchets dans un centre de traitement des ordures. Ce système pourrait également être intégré dans des caméras publiques ou d'autres dispositifs automatiques afin d'identifier les types de déchets présents dans des lieux publics. Pour ce faire, il est possible d'utiliser les réseaux de neurones convolutifs pour classifier les objets détectés. Cependant, plusieurs déchets ou un environnement complexe peut-être présent en même temps dans l'image, alors il est intéressant d'utiliser la segmentation pour diviser les régions d'intérêt. De plus, la limitation d'accès à une puissance informatique suffisante pour certains systèmes de tri des déchets, ainsi que le désir de rendre cette analyse mobile, soulève des questions sur la performance des différentes méthodologies de segmentation d'instance par rapport aux ressources utilisées lors de l'analyse.

1.2 Problématique

Plusieurs défis se posent actuellement pour la classification des déchets dans des environnements réels. En effet, des variations de lumière ou la présence d'autres éléments dans la scène peuvent compliquer la segmentation des objets. C'est pourquoi on cherche à comparer plusieurs approches afin de déterminer la solution la plus adaptée

à nos applications. Dans un premier temps, il sera essentiel d'évaluer l'efficacité des modèles, leur efficience, ainsi que leur performance en termes de ressources utilisées. De plus, dans le contexte où la solution serait déployée dans des lieux où la puissance informatique est limitée, il est crucial d'identifier quel modèle serait le moins gourmand en calcul. Pour cela, on intéresse à trois méthodologies : SOLOv2, Mask R-CNN et U-Net.

1.3 Objectif du projet

L'objectif principal de ce projet est de comparer trois méthodologies de segmentation d'images : SOLOv2, Mask R-CNN et U-Net dans le cadre d'une application de segmentation d'instance de déchets. L'analyse se concentrera sur l'efficacité (précision des résultats), l'efficience (rapidité de traitement) et l'utilisation des ressources, en particulier dans des environnements où les capacités de calcul sont limitées, comme les plateformes de tri de déchets ou les dispositifs mobiles. Ce projet vise à fournir une contribution en deux parties : premièrement, une analyse approfondie de ces trois approches dans ce contexte spécifique, et deuxièmement, une proposition de méthodologie et de critères d'évaluation pour déterminer la performance des différentes solutions. Enfin, des recommandations seront faites pour identifier la méthodologie la plus adaptée à ces applications concrètes.

- Fournir une analyse des trois méthodes SOLOv2, Mask R-CNN et U-Net dans le contexte de la détection de déchets.

- Proposer une méthodologie et des critères d'évaluation pour la segmentation en considérant l'efficacité, l'efficience et l'utilisation des ressources dans des environnements à ressources limitées.

- Fournir une analyse des trois méthodes dans le contexte de la détection de déchets et une recommander la meilleure méthodologie pour les environnements avec des ressources limitées.

Concernant la méthodologie, on utilise 2 backbone (ResNet-50 et 18) pour tous les tests afin de garantir une comparaison équitable et analyser la demande en ressources des modèles. La première étape consistera à effectuer une revue de littérature critique pour analyser les concepts des différentes méthodes de segmentation. Ensuite, pour évaluer la performance des méthodes dans notre mise en situation, nous procéderons en trois phases :

- **Efficacité** : on comparera les résultats des méthodes de segmentation avec le Ground Truth en analysant la superposition des deux masques (résultats obtenus et réalité).

- **Efficience** : On calculera la proportion de superposition des masques (si au moins 50% du masque généré par le modèle correspond à la réalité).

- **Utilisation des ressources** : On mesurera la quantité de ressources nécessaires pour chaque méthode, cet aspect sera affiné après une revue de la littérature.

Les jeux de données (datasets) seront **TACO** [1]. Enfin, sur la base de ces trois critères, nous déterminerons quelle méthodologie de segmentation offre les meilleures performances pour cette application.

1.4 Structure du document

Le document commencera par la présentation des architectures des modèles étudiés, on analysera en détail les trois modèles principaux utilisés, SOLOv2, Mask R-CNN, et U-Net, en examinant leurs fonctions de perte et leur processus d'entraînement. Ensuite on présentera la méthodologie expérimentale pour évaluer les performances, comprendre la base de données et définir les mesures de performances souhaitées. Par la suite, nous allons présenter les résultats et interpréter les résultats en mettant en avant leurs points forts et leurs limitations. Cette évaluation permettra de formuler une recommandation sur la technologie la plus performante et de proposer les prochaines étapes nécessaires pour atteindre notre objectif qui est d'identifier le modèle le mieux adapté à notre application.

2. SYNTHÈSE DES TECHNIQUES

2.1.1 SOLOV2

SOLOv2 est une méthode de détection et de segmentation d'objets que l'on utilisera dans ce projet. Contrairement aux modèles tels que Mask R-CNN, qui génèrent d'abord des propositions de boîtes englobantes avant d'appliquer la segmentation, SOLOv2 prédit

directement les masques de segmentation et les classes d'objets en utilisant la carte de caractéristiques. [2] Le modèle SOLOv2 se composent de deux parties principales : un réseau backbone, généralement un ResNet, qui extrait la carte des caractéristiques de l'image, et le module SOLO qui utilise ces caractéristiques pour générer des prédictions de masques de segmentation et de classes.

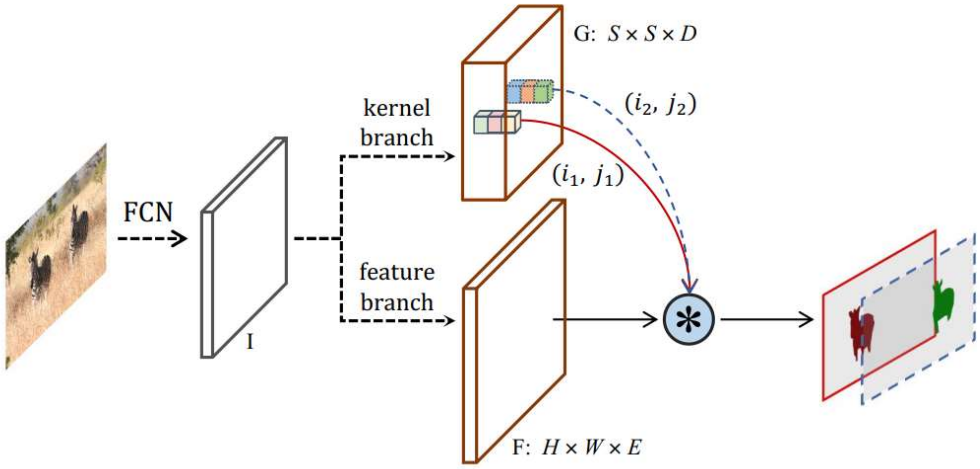

Figure 1 Représentation de l'architecture de SOLOv2 [2]

L'architecture de SOLOv2 est organisée en deux branches : une branche *kernel* et une branche *feature*. La branche *kernel* a pour rôle de générer dynamiquement les masques en utilisant les informations de la carte de caractéristiques, créant un masque spécifique pour chaque cellule de l'image. Cette approche dynamique adapte les masques aux objets en fonction de leur position et de leur taille dans l'image. Pour construire ces masques, on applique une fusion pyramidale [3] des caractéristiques, permettant une

sortie de masque de résolution 1/4 de l'image originale. Cette fusion est réalisée via des étapes de convolution, de normalisation par groupe et de fonction ReLU. [4]

Chaque étape intègre un processus de *upsampling* [5], atteignant ainsi une résolution de plus en plus fine. Une couche de convolution, suivie d'une normalisation et d'une activation ReLU, est appliquée après la sommation de toutes les cartes de caractéristiques fusionnées, produisant une représentation prête à être utilisée pour la création de masques de chaque objet. Une technique de binarisation, utilisant une fonction sigmoïde avec un seuil de 0,5, est ensuite appliquée pour convertir les valeurs en masque binaire. Les pixels ayant des valeurs égales ou supérieures à 0,5 sont considérés comme faisant partie de l'objet, tandis que les autres sont ignorés.

$$\mathrm{P}(o,i,j) = \sigma\big(f_{mask}(F)\big)$$

Finalement, la branche *kernel* est appliquée à la branche *feature* via pour générer le masque final.

$$M_{i,j} = G_{i,j} \circledast F$$

Une matrice NMS (Non-Maximum Suppression) est ensuite appliquée pour supprimer les masques redondants représentant le même objet ou des objets similaires. Cela permet de conserver uniquement les meilleurs masques pour chaque objet, en éliminant les duplications. Cette fonctionnalité est particulièrement utile dans notre application, car elle

permet de dénombrer précisément le nombre réel de déchets présents dans un environnement donné.

2.1.1.1 Les fonctions de perte

La fonction de perte, notée L, guide le modèle en ajustant ses paramètres pendant l'entraînement pour améliorer sa précision. Elle est définie par :

$$\mathrm{L} = \mathrm{Lcate} + \lambda \mathrm{Lmask}$$

où :

λ est un hyperparamètre pour équilibrer les deux termes de la perte.

La Focal Loss (Lcate) est utilisée pour gérer le déséquilibre entre les classes dans les images, où de nombreuses cellules de la grille contiennent des éléments sans objet. Elle est définie comme suit :

$$\mathrm{Lcate} = -\alpha(1-p)^{\gamma} log(p)$$

Dans l'équation p représente la probabilité de prédire la classe sans erreur et \alpha et ɣ sont des hyperparamètres ou a α est un facteur d'équilibrage des classes, introduit pour compenser le déséquilibre entre les classes positives et négatives. Ɣ est le paramètre de qui contrôle l'importance des échantillons bien classés par rapport aux échantillons mal classés.

Dans notre cas, ces hyperparamètres sont particulièrement pertinents. Par exemple, le paramètre α permet d'ajuster l'importance relative des catégories, ce qui corrige le déséquilibre entre les classes. Si les objets d'une catégorie spécifique sont rares, on peut utiliser une valeur élevée pour α afin d'augmenter l'impact de ces objets lors de l'apprentissage. Puisque nous essayons de détecter différents types de déchets dans des environnements réels, il sera essentiel d'ajuster ces hyperparamètres pour accentuer les déchets comme objets importants dans l'environnement, facilitant leur identification et améliorant les performances du modèle. [6]

Le dernier élément de fonction de coût est le Dice Loss. Elle est utilisée pour mesurer la similarité entre le masque prédit et le masque réel. Elle est particulièrement efficace pour la segmentation, car elle améliore la précision des contours de l'objet où la classe est déséquilibrée, comme dans notre cas lorsque les déchets ne sont pas nécessairement un objet majeur de la photo. Ce modèle permet de se concentrer sur les zones importantes et mieux détecter les petites régions ciblées. Elle est définie comme :

$$Lmask = 1 - \frac{2\sum_i y_i \hat{y}_i}{\sum_i y_i + \sum_i \hat{y}_i}$$

où :

yi est le masque réel et ŷ est le mask prédit.

La fonction est basée sur le coefficient de Dice, une mesure de similarité qui compare 2 ensembles en fonction de leur recouvrement. Elle est particulièrement efficace pour maximiser la similarité entre les pixels prédits et les pixels ciblés en favorisant le recouvrement optimal des 2 masques. [7]

2.1.2 Mask R-CNN

Mask R-CNN est un modèle de segmentation d'instances qu'on utilisera dans ce travail. Auparavant, nous avons exploré divers types d'architectures, notamment celles basées sur R-CNN et RPN, qui constituent des composantes essentielles de Mask R-CNN. Mask R-CNN repose sur le modèle Faster R-CNN, expliqué précédemment, qui génère des prédictions de classes d'objets (labels) et des boîtes englobantes (*bounding boxes*). Mask R-CNN y ajoute une troisième branche, qui applique un masque de segmentation à chaque objet détecté dans ces boîtes englobantes.[8]

L'architecture de Mask R-CNN peut être décomposée en trois branches principales :

Branche de classification : Prédis la classe de chaque objet détecté.

Branche de localisation : Génère les boîtes englobantes pour localiser les objets dans l'image.

Branche de segmentation : Produis un masque de segmentation pour chaque boîte englobante.

Comme nous avons déjà exploré les deux premières branches, nous nous concentrerons ici sur la branche de segmentation.

Le sous-réseau de segmentation reçoit les régions d'intérêt (RoIs) alignées grâce à l'opération *RoIAlign* [9], qui ajuste les pixels pour correspondre précisément aux objets détectés. Chaque RoI est ensuite transformé en une carte de caractéristiques de taille fixe et passe par plusieurs couches de convolution. Ces couches capturent les détails spatiaux des objets, tels que leurs contours et leurs bordures.

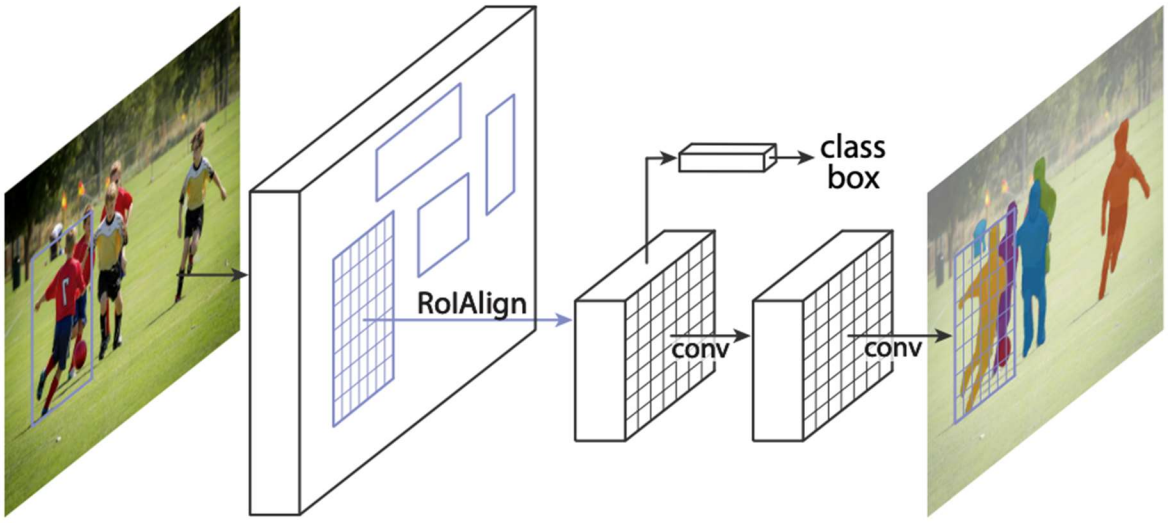

Figure 2 Représentation de l'architecture Mark R-CNN [8]

La sortie de ces couches de convolution est ensuite redimensionnée pour produire un masque binaire de prédiction pour chaque classe d'objets, habituellement en utilisant une fonction sigmoïde pour obtenir des valeurs entre 0 et 1. Le modèle génère des masques

indépendamment pour chaque classe, ce qui permet de prédire plusieurs masques correspondant aux différentes classes possibles. Cependant, seul le masque associé à la classe prédite est conservé pour chaque objet.

2.1.2.1 Fonctions de Perte

Pour entraîner Mask R-CNN, on utilise trois fonctions de perte distinctes correspondant aux trois branches, de manière à optimiser la classification, la localisation et la segmentation des objets.

$$\mathrm{Ltotal} = \mathrm{Lclass} + \mathrm{Lbbox} + \mathrm{Lmask}$$

Chaque terme de la fonction de perte peut être pondéré pour augmenter l'importance d'une tâche en particulier.

La fonction perte de la classification de Mask R-CNN utilise une fonction Cross Entropy, cette fonction compare la probabilité prédite pour chaque catégorie avec la catégorie réelle. Elle mesure la différence entre les prédictions du modèle et les étiquettes réelles en quantifiant l'incertitude.

$$\mathrm{Lclass} = \sum y \cdot log(p)$$

La fonction de perte de la branche de bounding box affine la position et la taille du cadre de délimitation prédit pour chaque objet. Elle est calculée uniquement pour les propositions de régions correctement classifiées, c'est-à-dire lorsque la prédiction correspond à l'étiquette de classe. Dans le modèle Mask R-CNN, on utilise une Smooth L1 Loss pour cette tâche. Cette fonction est particulièrement utile pour le réglage fin des bounding boxes, car initialement les cadres prédits peuvent être loin de leur position précise. La Smooth L1 Loss permet de modérer le gradient pour éviter que le modèle ne varie de manière excessive. Au fur et à mesure que l'apprentissage progresse et que les bounding boxes se rapproche de plus en plus de la vérité terrain (ground truth), le terme quadratique de la fonction permet d'affiner davantage la précision de la localisation. En d'autres termes, cette fonction de perte offre un bon équilibre entre sensibilité et stabilité pour le modèle.[10]

$$\text{SmoothL1t(x)} = \begin{cases} 0.5 \cdot x^2 & si\ |x| < 1 \\ |x| - 0.5 & sinon \end{cases}$$

Dans un contexte de segmentation :

$$\text{Lbbox} = \sum \text{SmoothL1(t\^i} - \text{ti)}$$

La perte de segmentation, Lmask, utilise une entropie croisée binaire pour chaque pixel du masque. Cette fonction de perte permet de minimiser l'erreur entre le masque prédit et le masque réel pour chaque pixel dans les régions d'intérêt. [11]

$$\text{Lmask} = -(y \cdot log(p) + (1 - y) \cdot log(1 - p))$$

Comme le modèle cherche à déterminer si chaque pixel appartient ou non au masque, l'entropie croisée mesure la distance entre la distribution prédite et la distribution réelle. Plus l'entropie croisée est faible, plus le modèle est proche de la réalité et plus la prédiction de chaque pixel dans le masque est précise.

2.1.3 U-Net

Le modèle **U-Net** est en fait le modèle de référence que l'on utilisera pour analyser la performance de tous les modèles dans ce projet. U-Net est un modèle populaire qui utilise une architecture basée sur les réseaux de neurones convolutionnels, particulièrement utilisée en segmentation d'image. La structure de base de U-Net est composée de deux parties, reposant sur une architecture encodeur-décodeur. [12]

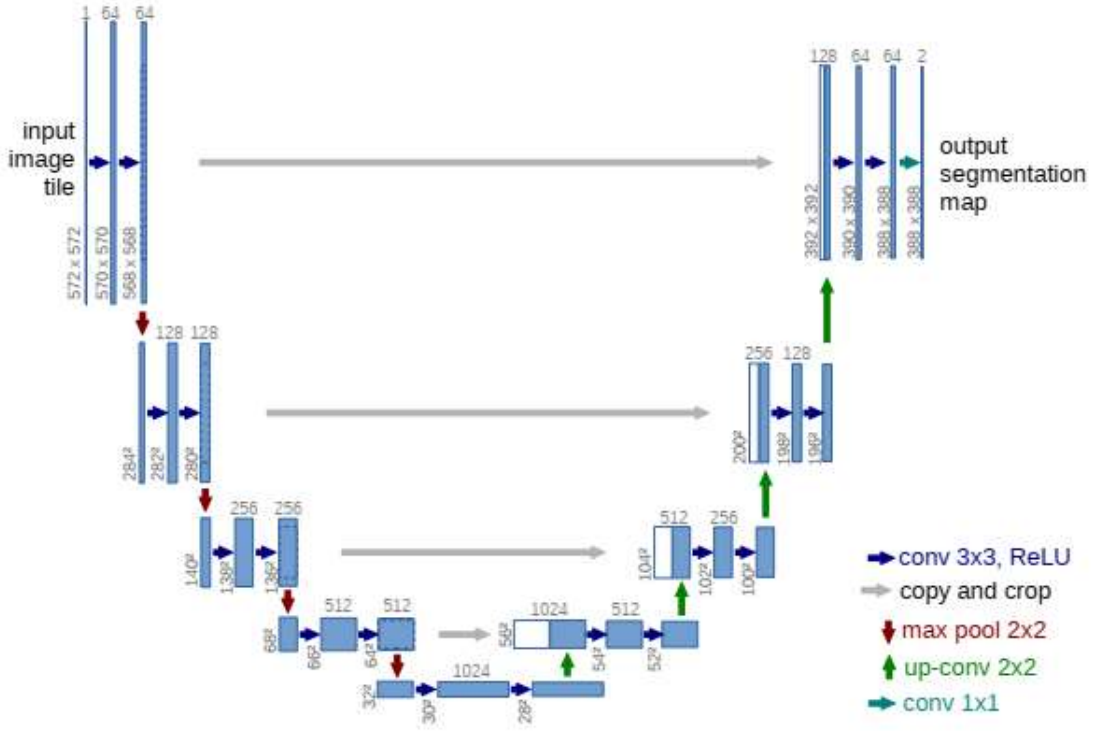

Figure 3 Représentation de U-NET [12]

L'encodeur est une série de couches de convolution qui réduit progressivement la taille de l'image et extrait les caractéristiques de plus bas niveau grâce à des filtres. Ensuite, le décodeur applique une série de couches de convolution et d'opérations de suréchantillonnage qui restaurent la résolution initiale de l'image. On obtient finalement une carte de caractéristiques segmentée ayant la même dimension que l'image d'entrée.

Une des particularités de U-Net est l'utilisation de sauts de connexion entre les couches de même niveau de l'encodeur et du décodeur. Cela permet de réutiliser les informations de bas niveau dans le processus de décodeur. Les sauts de connexion sont un concept

central dans l'architecture U-Net, car ils permettent de transférer des informations directement d'une couche de l'encodeur vers une couche correspondante du décodeur, en contournant plusieurs couches intermédiaires. Ceci est particulièrement utile dans les tâches de segmentation d'image, car cela aide à préserver les détails fins, essentiels pour obtenir une segmentation précise.

À chaque étape de l'encodeur, un saut de connexion est envoyé à l'étape correspondante du décodeur. Celui-ci copie la carte d'activation de la couche de l'encodeur et la concatène avec celle du décodeur. Cette fonctionnalité permet de préserver les détails des couches de l'encodeur, qui contiennent des informations telles que les contours et les textures des éléments, en les transférant directement au décodeur. Cela améliore ainsi la reconstruction des masques de segmentation.

Il est à noter qu'il est essentiel que ces opérations soient effectuées en parallèle avec leur étape correspondante dans le décodeur pour que le bloc de décodeur ait la même taille que celui de l'encodeur. Par exemple, si le bloc d'encodeur est de taille 128×128 et qu'après une étape de max-pooling il devient 64×64, cette taille est stockée pour un futur saut de connexion qui sera concaténé au décodeur lorsque celui-ci aura aussi une taille de 64×64.

La sortie de l'encodeur E et la sortie du décodeur D à un niveau donné, la concaténation exprimée de la façon suivante :

$$F = Concat(E, D)$$

Pour l'entrainement, U-NET utilise une fonction de perte basée sur l'entropie corsée binaire et le coefficient de Dice vue si dessus.

Finalement, le modèle U-Net sera un modèle essentiel, servant de référence pour évaluer les performances des autres modèles. Grâce à ses fonctionnalités, U-Net permet d'améliorer la qualité de la segmentation en aidant le modèle à mieux distinguer les frontières des objets et à reconnaître les structures plus fines.

3. MÉTHODOLOGIE EXPÉRIENTALE

Pour effectuer notre expérimentation, nous avons utilisé plusieurs bibliothèques implémentant les architectures nécessaires. MMDetection est un framework open source spécialisé dans la détection d'objets, développé par OpenMMLab. Ce framework est particulièrement utile dans notre cas, car il implémente des modèles comme Mask R-CNN et SoloV2, tout en offrant des outils pour l'entraînement, l'évaluation et le déploiement des modèles de détection d'objets. MMDetection repose sur une architecture modulaire qui permet d'ajuster facilement les composantes, telles que le backbone et les méthodes de post-traitement, afin de répondre à des besoins spécifiques.

Environnement

Toutes les spécifications détaillées de l'environnement se trouvent en annexe 3. Les composants essentiels incluent :

- CUDA Toolkit : version 11.7
- PyTorch : version 2.0.0
- MMCV : version 2.0.4

3.1 Protocole pour évaluer les performances

L'évaluation des performances a été réalisée selon plusieurs constantes. Tous les modèles utilisent deux types de backbones, ResNet-50 et ResNet-18, tous deux préentraînés sur le dataset COCO.

Un environnement Conda a été créé pour supporter les tests. Les paramètres d'entraînement utilisés sont les suivants et détaillé à l'annexe 4:

- Taux d'apprentissage: 0.0001
- Taille batch: 10
- Nombre d'époques: 30
- Nombre de workers: 1

3.2 Description de la base de données

Le dataset TACO (Trash Annotations in Context) est dédié à la segmentation des déchets dans des environnements variés. Ce dataset contient 1 500 images officielles, annotées au format COCO avec des masques de segmentation et des catégories correspondantes. La figure **X** présente les 28 super-catégories présentes dans le dataset. De plus, il inclut 60 sous-catégories, couvrant une large variété de matériaux disponible à l'annexe 3.

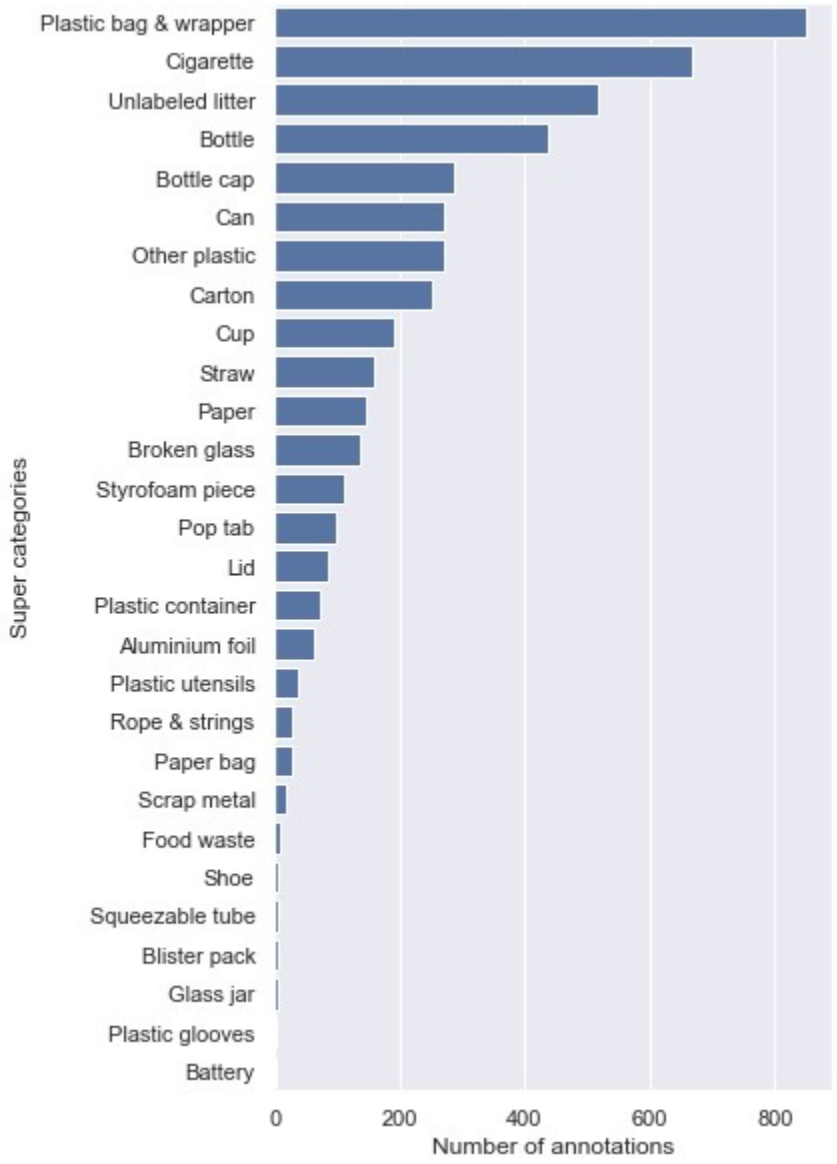

Figure 4: Distribution des images par classe principal

Certaines particularités de ce dataset en font une ressource de très bonne qualité. Premièrement, les images ont été annotées manuellement, ce qui garantit un haut niveau de précision des annotations. De plus, les scènes représentées dans le dataset sont

réalistes. Contrairement à d'autres datasets dédiés aux déchets, celui-ci propose une grande variété d'environnements, rendant l'apprentissage plus robuste pour des applications en conditions réelles.

La complexité contextuelle est également bien représentée. Plusieurs déchets peuvent être partiellement recouverts, déformés ou mélangés à d'autres objets. Comme illustré à la figure X, le dataset présente une grande variété d'arrière-plans, ce qui renforce sa pertinence pour des scénarios réels.

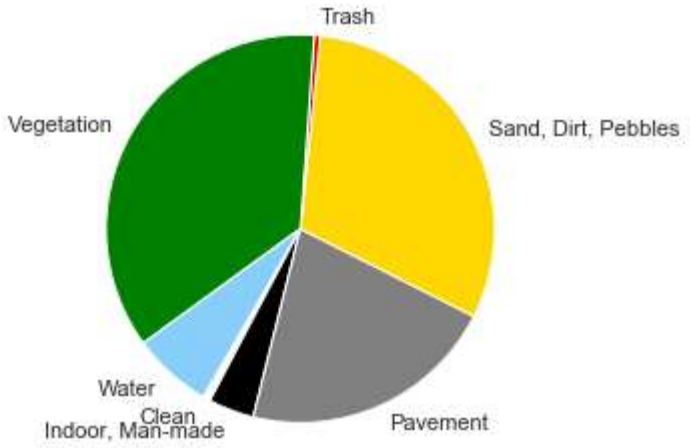

Figure 5: Distribution des environnements de chaque image

Cependant, certaines limites sont présentes, notamment la taille relativement réduite du dataset, malgré le grand nombre de classes disponibles. En annexe 3, nous mettons en évidence le déséquilibre des classes annotées. Ce déséquilibre pourrait être atténué par l'utilisation de techniques d'augmentation de données.

3.3 Mesures de performances

Pour évaluer la performance des modèles étudiés, nous utilisons deux types de métriques principales : des mesures de précision et des mesures d'efficacité énergétique.

Mesure de précision : mAP

La moyenne des précisions (mAP) est une métrique standard utilisée en vision par ordinateur pour évaluer la qualité des modèles de segmentation d'objets. Dans ce travail, nous utilisons le mAP 0.5 et 0.95, correspondant à la moyenne des valeurs de précision calculées à différents seuils d'intersection sur l'union (IoU). L'IoU mesure la correspondance entre la segmentation prédite et la segmentation de référence.

Les résultats sont compilés via les TensorBoards générés lors des inférences.

Mesure complémentaire : FLOPs et Efficacité énergétique

Un des objectifs de ce travail est de déterminer quelle architecture offre une solution viable pour une intégration dans un agent autonome. Ainsi, au-delà des performances en termes de précision, nous nous intéressons aux métriques liées à l'efficacité énergétique.

Le nombre de FLOPs est une mesure représentant la complexité computationnelle d’un modèle, exprimée par le nombre d'opérations nécessaires pour traiter les images

d'entrée. Cette métrique permet d’évaluer la viabilité énergétique des solutions proposées.

La librairie MMdetection offre une variété de fonctions d’analyse, dons get_flops.py qui permet d’avoir selon la grandeur d’image d’entrée le nombre de FLOPs et le nombre de Params.

4. RÉSULTATS

4.1 Limitation des résultats

Lors de l'expérimentation, nous avons rencontré de nombreux défis pour produire les résultats. Plusieurs outils utilisés, tels que MMDetection et TACO, ne sont plus activement maintenus. L'un des principaux problèmes récurrents concernait le manque de compatibilité des bibliothèques requises. Ces projets contiennent une quantité importante de code legacy et documentent rarement les versions exactes de PyTorch, Python ou des toolkits CUDA nécessaire pour utiliser ces technologies.

Par ailleurs, plusieurs erreurs ont été identifiées au niveau du dataset TACO. En effet, aucune maintenance ne semble avoir été effectuée depuis les quatre dernières années, ce qui a engendré des problèmes non résolus. Parmi eux, nous avons dû faire face à des erreurs de duplication d'identifiants, des photos manquantes, et des incohérences dans les annotations. Pour exploiter correctement les données, il a été nécessaire de résoudre manuellement ces problèmes, notamment en supprimant les doublons, en complétant les fichiers manquants et en rétrogradant certaines bibliothèques vers des versions compatibles.

En l'absence de résultats produits dans le cadre de ce projet, nous pouvons nous appuyer sur des expérimentations similaires réalisées dans la littérature. Par exemple, Xu & al. [13] ont utilisé SOLOv2 et Mask R-CNN sur un dataset incluant TACO. Ils ont déterminé

que Mask R-CNN présentait de meilleures performances en termes de mAP et de mAP50. Cependant, SOLOv2 s'est révélé plus intéressant en termes de rapidité d'exécution et de consommation de ressources.

5. CONCLUSIONS

En conclusion, les objectifs de ce travail ont été partiellement atteints. Nous avons fourni une analyse des trois méthodes SOLOv2, Mask R-CNN et U-Net dans le contexte de la détection de déchets. Une méthodologie et des critères d'évaluation pour la segmentation ont été proposés, prenant en compte l'efficacité, l'efficience et l'utilisation des ressources dans des environnements à ressources limitées.

Cependant, nous n'avons pas réussi à réaliser une analyse complète des trois méthodes dans le contexte de la détection de déchets ni à recommander la meilleure approche pour les environnements à ressources limitées, en raison des défis rencontrés lors de la production des résultats. Cette partie est essentielle pour déterminer la solution optimale pour notre application. Le manque de maintenance des bibliothèques, les problèmes liés au dataset, ainsi que la réduction des ressources informatiques disponibles ont eu un impact considérable sur l'avancement du projet.

C'est clairement une déception de ne pas avoir pu atteindre cet objectif. Avec du recul, le projet s'est avéré ambitieux compte tenu de nos connaissances préalables limitées. Un

projet de classification des instances aurait probablement été plus réaliste dans le temps imparti.

Pour la suite, il sera crucial de résoudre les problèmes liés au dataset TACO afin de pouvoir générer les résultats nécessaires. Par ailleurs, en approfondissant les recherches menées pour ce projet, il serait intéressant de tester d'autres backbones, tels que MobileNet, afin d'alléger la solution et de mieux répondre aux contraintes des environnements à ressources limitées.

A. ANNEXE 1 : CODE POUR LA PRODUCTION DES RÉSULTATS

Code pour Mask-RCNN

```
!pip install -U openmim
!mim install "mmengine>=0.7.0"
!mim install "mmcv>=2.0.0rc4"

!rm -rf mmdetection
!git clone https://github.com/open-mmlab/mmdetection.git
%cd mmdetection
!pip install -e .

!cd /TACO && pip install -r requirements.txt

!pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI

!python download.py
!python split_dataset.py --dataset_dir ../data

import torch, torchvision
print("torch version:", torch.__version__, "cuda:", torch.cuda.is_available())

import mmdet
print("mmdetection:", mmdet.__version__)

import mmcv
print("mmcv:", mmcv.__version__)

import mmengine
print("mmengine:", mmengine.__version__)

!mim   download   mmdet   --config   mask-rcnn_r50-caffe_fpn_ms-poly-3x_coco   --dest
./checkpoints
!mim download mmdet --config solov2_r50-caffe_fpn_ms-poly-3x_coco --dest ./checkpoints

import mmcv
import mmengine
from mmdet.apis import init_detector, inference_detector
```

```
from mmdet.utils import register_all_modules

config_file = 'configs/mask_rcnn/mask-rcnn_r50-caffe_fpn_ms-poly-3x_coco.py'
checkpoint_file = 'checkpoints/mask_rcnn_r50_caffe_fpn_mstrain-poly_3x_coco_bbox_mAP-
0.408__segm_mAP-0.37_20200504_163245-42aa3d00.pth'

register_all_modules()

model = init_detector(config_file, checkpoint_file, device='cuda:0')

img = mmcv.imread('SYS843/TACO/train/000002.jpg')
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 10))
plt.imshow(mmcv.bgr2rgb(img))
plt.show()

import mmengine
from pycocotools.coco import COCO

annotation_file = 'SYS843/TACO/train/annotation_coco.json'
coco = COCO(annotation_file)
categories = coco.loadCats(coco.getCatIds())
category_id_to_name = {cat['id']: cat['name'] for cat in categories}

for category_id, category_name in category_id_to_name.items():
    print(f"Category ID: {category_id}, Category Name: {category_name}")

from mmengine import Config
cfg       =       Config.fromfile('SYS843/mmdetection/configs/mask_rcnn/mask-rcnn_r50-
caffe_fpn_ms-poly-1x_coco.py')

cfg.metainfo = {
    'classes': list(category_id_to_name.values()),
    'palette': [(220, 20, 60)]
}

cfg.data_root = 'SYS843/TACO'
cfg.train_dataloader.dataset.ann_file = 'train/annotation_coco.json'
cfg.train_dataloader.dataset.data_root = cfg.data_root
cfg.train_dataloader.dataset.data_prefix.img = 'train/'
```

```
cfg.train_dataloader.dataset.metainfo = cfg.metainfo

cfg.val_dataloader.dataset.ann_file = 'val/annotation_coco.json'
cfg.val_dataloader.dataset.data_root = cfg.data_root
cfg.val_dataloader.dataset.data_prefix.img = 'val/'
cfg.val_dataloader.dataset.metainfo = cfg.metainfo

cfg.test_dataloader = cfg.val_dataloader

cfg.val_evaluator.ann_file = cfg.data_root + '/' + 'val/annotation_coco.json'
cfg.test_evaluator = cfg.val_evaluator

cfg.model.roi_head.bbox_head.num_classes = 1
cfg.model.roi_head.mask_head.num_classes = 1

cfg.load_from  =  'checkpoints/mask_rcnn_r50_caffe_fpn_mstrain-poly_3x_coco_bbox_mAP-
0.408__segm_mAP-0.37_20200504_163245-42aa3d00.pth'

cfg.work_dir = './tutorial_exps'
cfg.train_cfg.val_interval = 3
cfg.default_hooks.checkpoint.interval = 3
cfg.optim_wrapper.optimizer.lr = 0.02 / 8
cfg.default_hooks.logger.interval = 10

from mmengine.runner import set_random_seed
set_random_seed(0, deterministic=False)

cfg.visualizer.vis_backends.append({"type": 'TensorboardVisBackend'})

with     open(f'SYS843/mmdetection/configs/mask_rcnn/mask-rcnn_r50-caffe_fpn_ms-poly-
3x_balloon.py', 'w') as f:
    f.write(cfg.pretty_text)

img = mmcv.imread('./TACO/train/7178882742_f090f3ce56_k.jpg', channel_order='rgb')
checkpoint_file = 'TACO_exps/epoch_12.pth'

model = init_detector(cfg, checkpoint_file, device='cuda0')

new_result = inference_detector(model, img)
print(new_result)
```

B. ANNEXE 2 : GRAPHIQUE DESCRIPTIF DU DATASET TACO

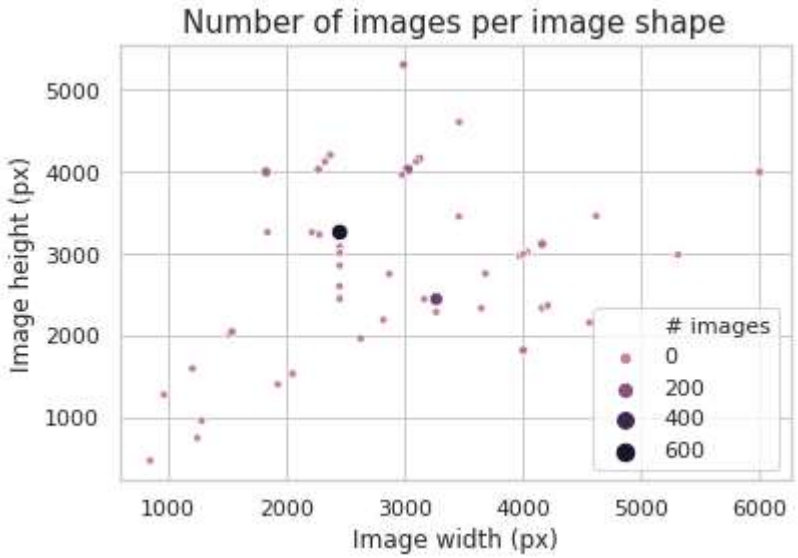

Figure 6: Distribution des dimensions des images TACO

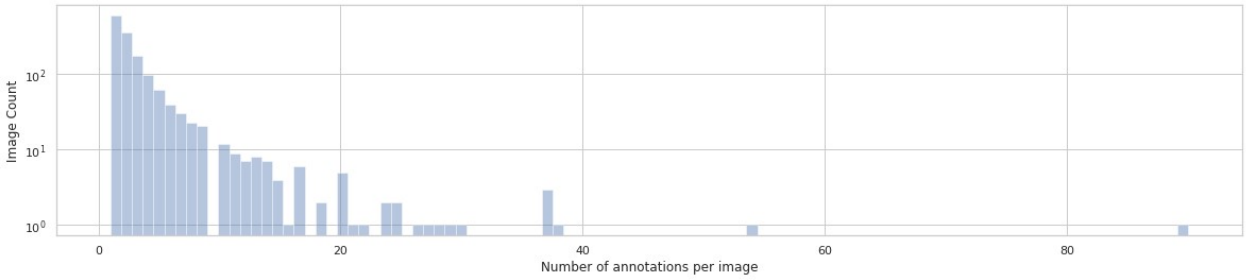

Figure 7: distribution des nombre d'annotation par image

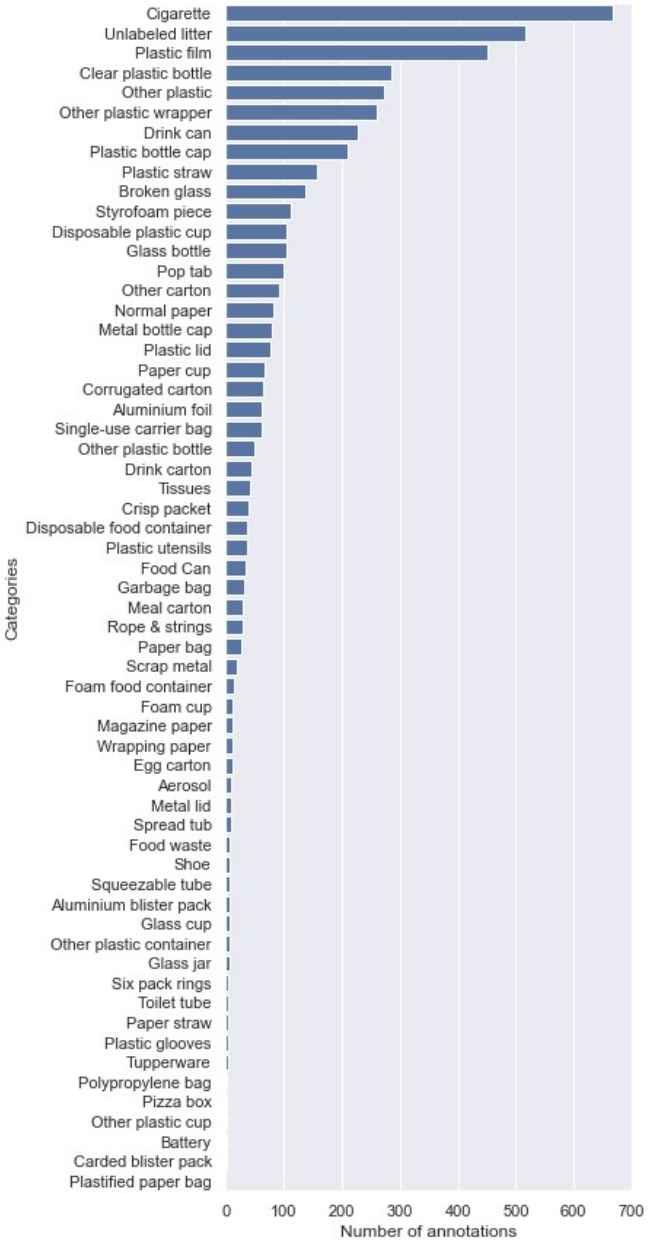

Figure 8: distribution des annotation par sous-catégorie

C. ANNEXE 3 : ÉCHANTILLONS D'IMAGE SEGMENTÉ PROVONANT DU DATASET TACO

Figure 9: Image d'instances

Figure 10: Image d'instances segmentées

Figure 11: Image d'instances segmentés avec la classe can

Figure 12: Image d'instances

Figure 13: Image d'instances segmentées

Figure 14: Image d'instances segmentées avec la classe plastique

D. ANNEXE 4 : ENVIRONNEMENT DE DÉVELOPPEMENT

```
name: mmLab
channels:
  - pytorch
  - nvidia
  - defaults
dependencies:
  - aiohappyeyeballs=2.4.0=py38haa95532_0
  - aiohttp=3.10.5=py38h827c3e9_0
  - aiosignal=1.2.0=pyhd3eb1b0_0
  - alabaster=0.7.12=pyhd3eb1b0_0
  - arrow=1.3.0=py38haa95532_0
  - astroid=3.2.4=py38haa95532_0
  - asttokens=2.0.5=pyhd3eb1b0_0
  - async-timeout=4.0.3=py38haa95532_0
  - asyncssh=2.17.0=py38haa95532_0
  - atomicwrites=1.4.0=py_0
  - attrs=24.2.0=py38haa95532_0
  - autopep8=2.0.4=pyhd3eb1b0_0
  - babel=2.11.0=py38haa95532_0
  - backcall=0.2.0=pyhd3eb1b0_0
  - beautifulsoup4=4.12.3=py38haa95532_0
  - binaryornot=0.4.4=pyhd3eb1b0_1
  - black=24.8.0=py38haa95532_0
  - blas=1.0=mkl
  - bleach=4.1.0=pyhd3eb1b0_0
  - brotli-python=1.0.9=py38hd77b12b_8
  - ca-certificates=2024.11.26=haa95532_0
  - certifi=2024.8.30=py38haa95532_0
  - cffi=1.17.1=py38h827c3e9_0
  - chardet=4.0.0=py38haa95532_1003
  - charset-normalizer=3.3.2=pyhd3eb1b0_0
  - click=8.1.7=py38haa95532_0
  - cloudpickle=3.0.0=py38haa95532_0
  - colorama=0.4.6=py38haa95532_0
  - comm=0.2.1=py38haa95532_0
  - cookiecutter=2.6.0=py38haa95532_0
  - cuda-cccl=12.6.77=0
  - cuda-cccl_win-64=12.6.77=0
  - cuda-cudart=11.7.99=0
  - cuda-cudart-dev=11.7.99=0
  - cuda-cupti=11.7.101=0
  - cuda-libraries=11.7.1=0
  - cuda-libraries-dev=11.7.1=0
  - cuda-nvrtc=11.7.99=0
  - cuda-nvrtc-dev=11.7.99=0
  - cuda-nvtx=11.7.91=0
  - cuda-runtime=11.7.1=0
  - cuda-version=12.6=3
  - debugpy=1.6.7=py38hd77b12b_0
  - decorator=5.1.1=pyhd3eb1b0_0
  - defusedxml=0.7.1=pyhd3eb1b0_0
  - deprecated=1.2.13=py38haa95532_0
  - diff-match-patch=20200713=pyhd3eb1b0_0
  - dill=0.3.8=py38haa95532_0
  - docstring-to-markdown=0.11=py38haa95532_0
  - docutils=0.18.1=py38haa95532_3
  - executing=0.8.3=pyhd3eb1b0_0
  - flake8=7.1.1=py38haa95532_0
  - freetype=2.12.1=ha860e81_0
  - frozenlist=1.4.0=py38h2bbff1b_0
  - gmpy2=2.1.2=py38h7f96b67_0
  - icu=73.1=h6c2663c_0
  - idna=3.7=py38haa95532_0
  - imagesize=1.4.1=py38haa95532_0
  - importlib_metadata=7.0.1=hd3eb1b0_0
  - importlib_resources=6.4.0=py38haa95532_0
  - inflection=0.5.1=py38haa95532_0
  - intel-openmp=2023.1.0=h59b6b97_46320
  - intervaltree=3.1.0=pyhd3eb1b0_0
  - ipykernel=6.29.5=py38haa95532_0
  - ipython=8.12.2=py38haa95532_0
  - isort=5.13.2=py38haa95532_0
  - jaraco.classes=3.2.1=pyhd3eb1b0_0
  - jedi=0.19.1=py38haa95532_0
  - jellyfish=1.0.1=py38h36a85e1_0
  - jinja2=3.1.4=py38haa95532_0
  - jpeg=9e=h827c3e9_3
  - jsonschema=4.23.0=py38haa95532_0
  - jsonschema-specifications=2023.7.1=py38haa95532_0
  - jupyter_client=8.6.0=py38haa95532_0
  - jupyter_core=5.7.2=py38haa95532_0
  - jupyterlab_pygments=0.2.2=py38haa95532_0
  - keyring=24.3.1=py38haa95532_0
  - krb5=1.20.1=h5b6d351_0
  - lcms2=2.12=h83e58a3_0
  - lerc=3.0=hd77b12b_0
  - libclang=14.0.6=default_hb5a9fac_1
  - libclang13=14.0.6=default_h8e68704_1
  - libcublas=11.10.3.66=0
  - libcublas-dev=11.10.3.66=0
  - libcufft=10.7.2.124=0
  - libcufft-dev=10.7.2.124=0
  - libcurand=10.3.7.77=0
  - libcurand-dev=10.3.7.77=0
  - libcusolver=11.4.0.1=0
  - libcusolver-dev=11.4.0.1=0
  - libcusparse=11.7.4.91=0
  - libcusparse-dev=11.7.4.91=0
  - libdeflate=1.17=h2bbff1b_1
  - libffi=3.4.4=hd77b12b_1
  - libnpp=11.7.4.75=0
  - libnpp-dev=11.7.4.75=0
  - libnvjpeg=11.8.0.2=0
  - libnvjpeg-dev=11.8.0.2=0
  - libpng=1.6.39=h8cc25b3_0
  - libpq=17.0=h70ee33d_0
  - libsodium=1.0.18=h62dcd97_0
  - libspatialindex=1.9.3=h6c2663c_0
  - libtiff=4.5.1=hd77b12b_0
  - libuv=1.48.0=h827c3e9_0
  - libwebp-base=1.3.2=h3d04722_1
  - lz4-c=1.9.4=h2bbff1b_1
  - markupsafe=2.1.3=py38h2bbff1b_0
  - matplotlib-inline=0.1.6=py38haa95532_0
  - mccabe=0.7.0=pyhd3eb1b0_0
  - mistune=2.0.4=py38haa95532_0
```

- mkl=2023.1.0=h6b88ed4_46358
- mkl-service=2.4.0=py38h2bbff1b_1
- mkl_fft=1.3.8=py38h2bbff1b_0
- mkl_random=1.2.4=py38h59b6b97_0
- more-itertools=10.3.0=py38haa95532_0
- mpc=1.1.0=h7edee0f_1
- mpfr=4.0.2=h62dcd97_1
- mpir=3.0.0=hec2e145_1
- mpmath=1.3.0=py38haa95532_0
- multidict=6.0.4=py38h2bbff1b_0
- mypy_extensions=1.0.0=py38haa95532_0
- nbclient=0.8.0=py38haa95532_0
- nbconvert=7.16.4=py38haa95532_0
- nbformat=5.10.4=py38haa95532_0
- nest-asyncio=1.6.0=py38haa95532_0
- networkx=3.1=py38haa95532_0
- numpydoc=1.5.0=py38haa95532_0
- openjpeg=2.5.2=hae555c5_0
- openssl=3.0.15=h827c3e9_0
- pandocfilters=1.5.0=pyhd3eb1b0_0
- parso=0.8.3=pyhd3eb1b0_0
- pathspec=0.10.3=py38haa95532_0
- pexpect=4.8.0=pyhd3eb1b0_3
- pickleshare=0.7.5=pyhd3eb1b0_1003
- pillow=10.4.0=py38h827c3e9_0
- pip=24.2=py38haa95532_0
- pkgutil-resolve-name=1.3.10=py38haa95532_1
- pluggy=1.0.0=py38haa95532_1
- ply=3.11=py38_0
- prompt-toolkit=3.0.43=py38haa95532_0
- psutil=5.9.0=py38h2bbff1b_0
- ptyprocess=0.7.0=pyhd3eb1b0_2
- pure_eval=0.2.2=pyhd3eb1b0_0
- pycodestyle=2.12.1=py38haa95532_0
- pydocstyle=6.3.0=py38haa95532_0
- pyflakes=3.2.0=py38haa95532_0
- pygithub=2.4.0=py38haa95532_0
- pyjwt=2.8.0=py38haa95532_0
- pylint=3.2.7=py38haa95532_0
- pylint-venv=3.0.3=py38haa95532_0
- pyls-spyder=0.4.0=pyhd3eb1b0_0
- pynacl=1.5.0=py38h8cc25b3_0
- pyqt=5.15.10=py38hd77b12b_0
- pyqt5-sip=12.13.0=py38h2bbff1b_0
- pyqtwebengine=5.15.10=py38hd77b12b_0
- pysocks=1.7.1=py38haa95532_0
- python=3.8.20=h8205438_0
- python-dateutil=2.9.0post0=py38haa95532_2
- python-fastjsonschema=2.16.2=py38haa95532_0
- python-lsp-black=2.0.0=py38haa95532_0
- python-lsp-jsonrpc=1.1.2=pyhd3eb1b0_0
- python-lsp-server=1.12.0=py38h9909e9c_0
- python-slugify=5.0.2=pyhd3eb1b0_0
- pytoolconfig=1.2.6=py38haa95532_0
- pytorch=2.0.0=py3.8_cuda11.7_cudnn8_0
- pytorch-cuda=11.7=h16d0643_5
- pytorch-mutex=1.0=cuda
- pyuca=1.2=py38haa95532_1
- pywin32-ctypes=0.2.2=py38haa95532_0
- pyyaml=6.0.2=py38h827c3e9_0
- pyzmq=25.1.2=py38hd77b12b_0
- qdarkstyle=3.2.3=pyhd3eb1b0_0
- qstylizer=0.2.2=py38haa95532_0
- qt-main=5.15.2=h19c9488_11
- qt-webengine=5.15.9=h5bd16bc_7
- qtawesome=1.3.1=py38haa95532_0
- qtconsole=5.6.0=py38haa95532_0
- qtpy=2.4.1=py38haa95532_0
- referencing=0.30.2=py38haa95532_0
- rope=1.12.0=py38haa95532_0
- rpds-py=0.10.6=py38h062c2fa_0
- rtree=1.0.1=py38h2eaa2aa_0
- sip=6.7.12=py38hd77b12b_0
- six=1.16.0=pyhd3eb1b0_1
- snowballstemmer=2.2.0=pyhd3eb1b0_0
- sortedcontainers=2.4.0=pyhd3eb1b0_0
- soupsieve=2.5=py38haa95532_0
- sphinx=5.0.2=py38haa95532_0
- sphinxcontrib-applehelp=1.0.2=pyhd3eb1b0_0
- sphinxcontrib-devhelp=1.0.2=pyhd3eb1b0_0
- sphinxcontrib-htmlhelp=2.0.0=pyhd3eb1b0_0
- sphinxcontrib-jsmath=1.0.1=pyhd3eb1b0_0
- sphinxcontrib-qthelp=1.0.3=pyhd3eb1b0_0
- sphinxcontrib-serializinghtml=1.1.5=pyhd3eb1b0_0
- spyder=6.0.1=py38haa95532_0
- spyder-kernels=3.0.0=py38h9909e9c_0
- sqlite=3.45.3=h2bbff1b_0
- stack_data=0.2.0=pyhd3eb1b0_0
- superqt=0.6.7=py38h9909e9c_0
- sympy=1.13.2=py38haa95532_0
- tbb=2021.8.0=h59b6b97_0
- text-unidecode=1.3=pyhd3eb1b0_0
- textdistance=4.2.1=pyhd3eb1b0_0
- three-merge=0.1.1=pyhd3eb1b0_0
- tinycss2=1.2.1=py38haa95532_0
- tomlkit=0.13.2=py38haa95532_0
- tornado=6.4.1=py38h827c3e9_0
- traitlets=5.14.3=py38haa95532_0
- typing_extensions=4.11.0=py38haa95532_0
- ujson=5.10.0=py38h5da7b33_0
- unidecode=1.3.8=py38haa95532_0
- vc=14.40=h2eaa2aa_1
- vs2015_runtime=14.40.33807=h98bb1dd_1
- watchdog=4.0.1=py38haa95532_0
- webencodings=0.5.1=py38_1
- whatthepatch=1.0.2=py38haa95532_0
- wheel=0.44.0=py38haa95532_0
- win_inet_pton=1.1.0=py38haa95532_0
- wrapt=1.14.1=py38h2bbff1b_0
- xz=5.4.6=h8cc25b3_1
- yaml=0.2.5=he774522_0
- yarl=1.11.0=py38h827c3e9_0
- zeromq=4.3.5=hd77b12b_0
- zipp=3.20.2=py38haa95532_0
- zlib=1.2.13=h8cc25b3_1
- zstd=1.5.6=h8880b57_0
- pip:
  - absl-py==2.1.0
  - addict==2.4.0
  - albucore==0.0.17
  - albumentations==1.3.1
  - aliyun-python-sdk-core==2.16.0
  - aliyun-python-sdk-kms==2.16.5
  - annotated-types==0.7.0

- cachetools==5.5.0
- contourpy==1.1.1
- crcmod==1.7
- cryptography==44.0.0
- cycler==0.12.1
- eval-type-backport==0.2.0
- filelock==3.14.0
- fonttools==4.55.0
- google-auth==2.36.0
- google-auth-oauthlib==1.0.0
- grpcio==1.68.0
- imageio==2.35.1
- importlib-metadata==8.5.0
- importlib-resources==6.4.5
- jmespath==0.10.0
- joblib==1.4.2
- kiwisolver==1.4.7
- lazy-loader==0.4
- markdown==3.7
- markdown-it-py==3.0.0
- matplotlib==3.7.5
- mdurl==0.1.2
- mmcv==2.0.1
- mmdet==3.3.0
- mmengine==0.10.5
- model-index==0.1.11
- numpy==1.24.4
- oauthlib==3.2.2
- opencv-python==4.10.0.84
- opencv-python-headless==4.10.0.84
- opendatalab==0.0.10
- openmim==0.3.9
- openxlab==0.1.2
- ordered-set==4.1.0
- oss2==2.17.0
- packaging==24.2
- pandas==2.0.3
- platformdirs==4.3.6
- prettytable==3.11.0
- protobuf==5.29.0
- pyasn1==0.6.1
- pyasn1-modules==0.4.1
- pycocotools==2.0.7
- pycparser==2.22
- pycryptodome==3.21.0
- pydantic==2.10.2
- pydantic-core==2.27.1
- pygments==2.18.0
- pyparsing==3.1.4
- pytz==2023.4
- pywavelets==1.4.1
- pywin32==308
- qudida==0.0.4
- regex==2024.11.6
- requests==2.28.2
- requests-oauthlib==2.0.0
- rich==13.4.2
- rsa==4.9
- scikit-image==0.21.0
- scikit-learn==1.3.2
- scipy==1.10.1
- setuptools==60.2.0
- shapely==2.0.6
- tabulate==0.9.0
- tensorboard==2.14.0
- tensorboard-data-server==0.7.2
- termcolor==2.4.0
- terminaltables==3.1.10
- threadpoolctl==3.5.0
- tifffile==2023.7.10
- tomli==2.2.1
- torchaudio==2.0.0
- torchvision==0.15.0
- tqdm==4.65.2
- typing-extensions==4.12.2
- tzdata==2024.2
- urllib3==1.26.20
- wcwidth==0.2.13
- werkzeug==3.0.6
- yapf==0.43.0

prefix:

6. BIBLIOGRAPHIE

[1] “Papers with Code - TACO Dataset.” Accessed: Nov. 22, 2024. [Online]. Available: https://paperswithcode.com/dataset/taco

[2] X. Wang, R. Zhang, T. Kong, L. Li, and C. Shen, “SOLOv2: Dynamic and Fast Instance Segmentation”.

[3] M. Oršić and S. Šegvić, “Efficient semantic segmentation with pyramidal fusion,” *Pattern Recognit.*, vol. 110, p. 107611, Feb. 2021, doi: 10.1016/j.patcog.2020.107611.

[4] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” Feb. 07, 2019, *arXiv*: arXiv:1803.08375. doi: 10.48550/arXiv.1803.08375.

[5] S. Song, F. Zhong, T. Wang, X. Qin, and C. Tu, “Guided Linear Upsampling,” Jul. 13, 2023, *arXiv*: arXiv:2307.09582. doi: 10.48550/arXiv.2307.09582.

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” Feb. 07, 2018, *arXiv*: arXiv:1708.02002. doi: 10.48550/arXiv.1708.02002.

[7] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, “Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations,” Jul. 14, 2017, *arXiv*: arXiv:1707.03237. doi: 10.48550/arXiv.1707.03237.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” Jan. 24, 2018, *arXiv*: arXiv:1703.06870. doi: 10.48550/arXiv.1703.06870.

[9] “Semantic-Aware Transformation-Invariant RoI Align.” Accessed: Oct. 27, 2024. [Online]. Available: https://arxiv.org/html/2312.09609v1

[10] C. Liu *et al.*, “Adaptive Smooth L1 Loss: A Better Way to Regress Scene Texts with Extreme Aspect Ratios,” in *2021 IEEE Symposium on Computers and Communications (ISCC)*, Sep. 2021, pp. 1–7. doi: 10.1109/ISCC53001.2021.9631466.

[11] Z. Zhang and M. R. Sabuncu, “Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels,” Nov. 29, 2018, *arXiv*: arXiv:1805.07836. doi: 10.48550/arXiv.1805.07836.

[12] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” May 18, 2015, *arXiv*: arXiv:1505.04597. doi: 10.48550/arXiv.1505.04597.

[13] N. Xu, J. Liao, Q. Meng, and W. Song, “Garbage Segmentation and Attribute Analysis by Robotic Dogs,” Apr. 28, 2024, *arXiv*: arXiv:2404.18112. doi: 10.48550/arXiv.2404.18112.