Projet 8 : Traitement des Images pour le Système Embarqué d'une Voiture Autonome

Résumé

Ce projet vise à développer une solution de segmentation d'images adaptée aux systèmes embarqués des véhicules autonomes de Future Vision Transport. En utilisant des variantes de l'architecture U-Net intégrant des modèles pré-entraînés comme VGG16, VGG19 et ResNet50, nous avons optimisé les performances de segmentation pour identifier des objets critiques tels que les piétons, les véhicules et les panneaux de signalisation. Les performances des modèles ont été évaluées à l'aide de métriques standard, et une API de prédiction ainsi qu'une application web ont été développées pour faciliter l'utilisation des résultats de la segmentation. Les résultats montrent une amélioration notable des performances grâce à l'utilisation de modèles pré-entraînés, tout en identifiant des pistes d'amélioration pour les futurs travaux.

Sommaire

Introduction

La segmentation d'images est un domaine crucial de la vision par ordinateur, notamment dans le contexte des systèmes embarqués pour les véhicules autonomes. Future Vision Transport, entreprise spécialisée dans la conception de ces systèmes, s'attache à développer des solutions avancées pour assurer la sécurité et l'efficacité des véhicules autonomes.

En tant qu'ingénieur en intelligence artificielle au sein de l'équipe R&D de cette entreprise, ma responsabilité principale réside dans la conception d'un modèle de segmentation d'images adapté à l'intégration fluide dans le système embarqué. Mon travail s'inscrit dans une chaîne de traitement où les images sont acquises en temps réel, traitées, segmentées (c'est ma tâche spécifique), puis utilisées pour prendre des décisions.

Le processus de segmentation d'images revêt une importance capitale dans cette chaîne, car il permet de distinguer et d'identifier différents éléments présents dans l'environnement du véhicule autonome, tels que les piétons, les véhicules, les panneaux de signalisation, etc. Cette distinction est essentielle pour que le système puisse prendre des décisions pertinentes et sécuritaires.

Dans ce cadre, je me suis engagé à répondre aux besoins spécifiques de Franck et Laura, respectivement responsables du traitement des images et du système de décision. Franck nécessite un jeu de données approprié pour l'entraînement du modèle de segmentation, tandis que Laura souhaite une interface simple et efficace pour utiliser les résultats de la segmentation dans le système de décision.

Ainsi, mon plan d'action comprend l'entraînement d'un modèle de segmentation sur les 8 catégories principales, en prenant en compte les contraintes et les besoins exprimés par Franck. De plus, je vais concevoir une API de prédiction, comme le souhaite Laura, qui permettra une utilisation aisée des résultats de la segmentation. Enfin, une application web sera développée pour présenter de manière conviviale les résultats de la segmentation, offrant ainsi une interface intuitive pour tester l'API et visualiser les images et les masques associés.

En résumé, mon objectif est de fournir une solution complète et intégrée pour la segmentation d'images, répondant aux exigences techniques et fonctionnelles de Future Vision Transport, dans le but ultime d'améliorer la sécurité et les performances des véhicules autonomes.

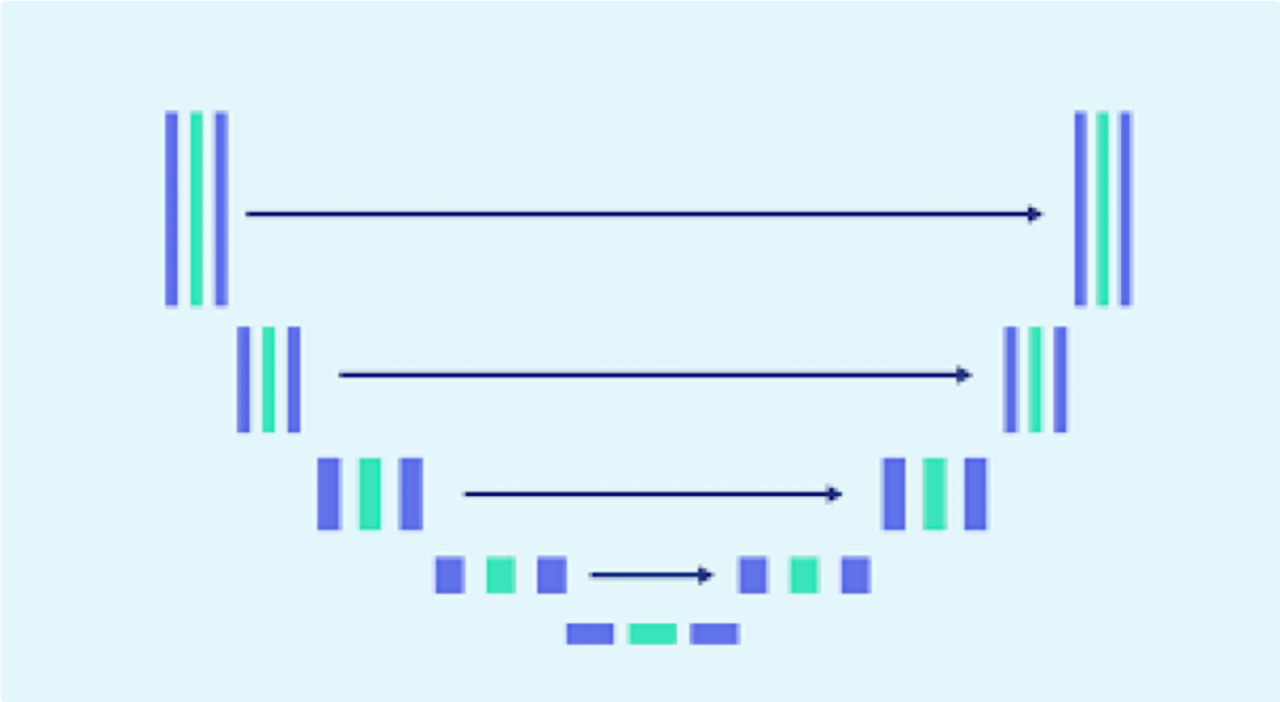
État de l'art

La segmentation d'images, en tant que tâche fondamentale de la vision par ordinateur, a vu l'émergence et l'évolution de diverses approches au fil des années. Dans cette section, nous présenterons un aperçu des différentes méthodes de segmentation d'images, en mettant particulièrement l'accent sur l'architecture U-Net et ses variantes.

Approches de segmentation d'images

La segmentation d'images peut être abordée de différentes manières, notamment par seuillage, regroupement de régions, méthodes basées sur les contours, et plus récemment, par l'utilisation de réseaux de neurones profonds. Parmi les méthodes traditionnelles, les techniques de seuillage et de regroupement de régions ont été largement utilisées pour segmenter les images en identifiant des régions homogènes en termes de couleur, de texture ou d'autres caractéristiques. Cependant, ces approches peuvent être limitées dans leur capacité à segmenter des objets complexes et à gérer des variations d'éclairage et de fond.

Architecture U-Net



L'architecture U-Net, introduite par Ronneberger et al. en 2015, a révolutionné le domaine de la segmentation d'images grâce à sa capacité à capturer efficacement les informations contextuelles tout en conservant une précision de localisation élevée. Cette architecture tire son nom de sa forme en U, constituée d'un encodeur (ou réseau de descente) suivi d'un décodeur (ou réseau de montée). L'encodeur est conçu pour extraire progressivement les caractéristiques de haut niveau de l'image, tandis que le décodeur reconstruit l'image segmentée à partir de ces caractéristiques en appliquant une upsampling et en incorporant des connexions résiduelles pour améliorer la localisation des objets.

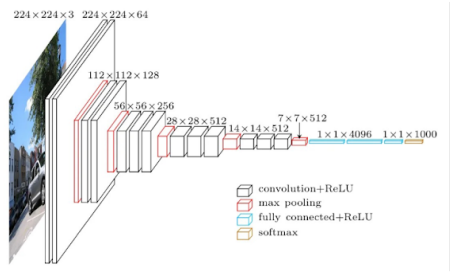
Variantes de l'architecture U-Net

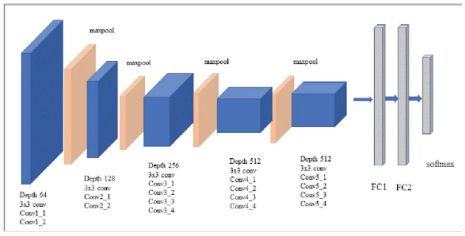
Nous avons adopté une variante de l'architecture U-Net dans notre travail, en intégrant des modèles pré-entraînés tels que VGG16, VGG19 et ResNet50 comme encodeurs dans notre réseau. Cette approche nous permet de bénéficier des caractéristiques visuelles déjà apprises par ces modèles sur le jeu de données ImageNet, améliorant ainsi les performances de segmentation. Plus précisément, notre architecture comprend plusieurs blocs de convolution pour capturer les caractéristiques de l'image, des blocs de pooling pour réduire la dimensionnalité, et des connexions de saut pour faciliter la propagation de l'information dans le réseau.

Fonctions définies dans le code

Pour mettre en œuvre notre architecture U-Net et ses variantes, nous avons défini plusieurs fonctions clés, notamment des blocs de convolution, des blocs d'encodeur et de décodeur, ainsi que des fonctions de construction de modèles spécifiques à chaque architecture. Ces fonctions sont conçues pour faciliter le processus de développement et d'expérimentation, tout en assurant la modularité et la réutilisabilité du code.

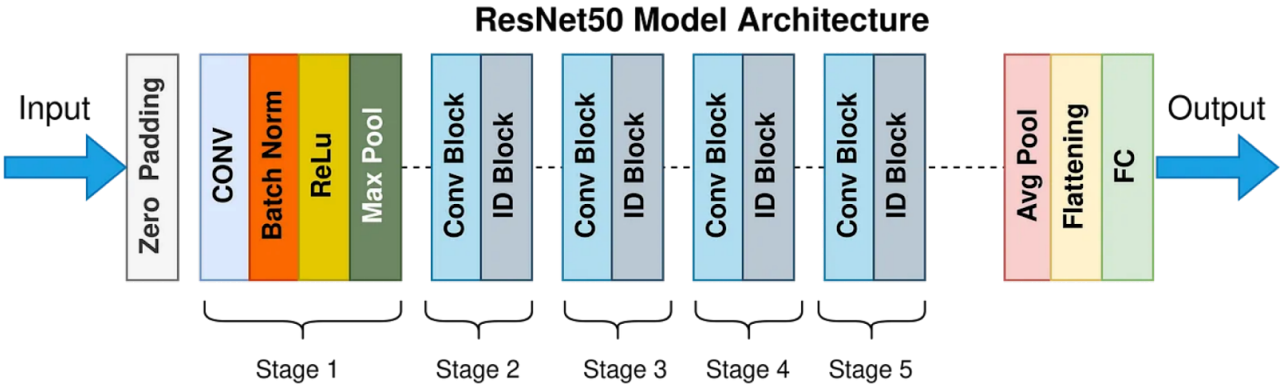
VGG16 et VGG19





Ces deux modèles ont été introduits par Simonyan et Zisserman en 2014. Ils se distinguent par leur architecture profonde et homogène, composée de plusieurs couches de convolution suivies de couches de pooling et de couches entièrement connectées. VGG16 contient 16 couches de convolution et de pooling, tandis que VGG19 en contient 19. Ces modèles ont été pré-entraînés sur le vaste ensemble de données ImageNet, ce qui leur permet de capturer efficacement les caractéristiques visuelles de divers objets et scènes.

ResNet50



Ce modèle, introduit par He et al. en 2015, a marqué une avancée significative en introduisant des connexions résiduelles, permettant de former des réseaux encore plus profonds avec une meilleure stabilité de l'entraînement. ResNet50 comprend 50 couches au total, avec des blocs résiduels qui facilitent le flux d'informations à travers le réseau. Comme VGG16 et VGG19, ResNet50 a été pré-entraîné sur ImageNet, offrant ainsi une représentation riche des caractéristiques visuelles.

Modèles et Architecture

Nous avons utilisé une variante de l'architecture U-Net, en intégrant des modèles pré-entraînés tels que VGG16, VGG19 et ResNet50 comme encodeur dans notre réseau. Ces modèles pré-entraînés ont été formés sur ImageNet, ce qui nous permet de bénéficier de caractéristiques visuelles déjà apprises.

L'architecture U-Net est composée d'un encodeur (réseau de descente) et d'un décodeur (réseau de montée). L'encodeur est utilisé pour capturer le contexte dans l'image, tandis que le décodeur permet de localiser précisément l'objet dans l'image.

Voici une description détaillée des fonctions définies dans le code :

- `conv_block(input, num_filters)` : Cette fonction définit un bloc de convolution qui comprend une couche de convolution, une normalisation par lots et une activation ReLU.
- `encoder_block(input, num_filters)` : Cette fonction définit un bloc d'encodeur dans l'architecture U-Net. Il comprend un bloc de convolution et une couche de pooling max.
- `decoder_block(input, skip_features, num_filters)` : Cette fonction définit un bloc de décodeur dans l'architecture U-Net. Il comprend une couche de transposition de convolution, une concaténation avec les caractéristiques de saut de l'encodeur correspondant et un bloc de convolution.
- `build_mini_unet(input_shape, n_classes)` : Cette fonction construit un modèle Mini U-Net pour la segmentation d'images. Il comprend deux blocs d'encodeur, un pont (qui est un bloc de convolution) et deux blocs de décodeur.
- `build_vgg16_unet(input_shape, nb_classes)` : Cette fonction construit un modèle U-Net basé sur VGG16 pour la segmentation d'images. Il utilise les caractéristiques de sortie de certaines couches du modèle VGG16 pré-entraîné comme caractéristiques de saut pour les blocs de décodeur.
- `build_resnet50(input_shape, nb_classes)` : Cette fonction construit un modèle U-Net basé sur ResNet50 pour la segmentation d'images. Il utilise les caractéristiques de sortie de certaines couches du modèle ResNet50 pré-entraîné comme caractéristiques de saut pour les blocs de décodeur.
- `build_vgg19_unet(input_shape, nb_classes)` : Cette fonction construit un modèle U-Net basé sur VGG19 pour la segmentation d'images. Il utilise les caractéristiques de sortie de certaines couches du modèle VGG19 pré-entraîné comme caractéristiques de saut pour les blocs de décodeur.

Dans toutes ces architectures, l'encodeur capture progressivement le contexte dans l'image tandis que le décodeur permet une localisation précise en utilisant les connexions de saut de l'encodeur au décodeur.

Générateur de données

Le `DataGenerator` est un élément crucial de notre pipeline pour préparer les données d'entraînement des modèles de segmentation d'images. Voici une description détaillée des fonctionnalités implémentées dans ce générateur :

- **Initialisation**: La méthode `__init__` initialise le générateur de données avec les paramètres nécessaires tels que les chemins des images et des masques, les dimensions des images, le nombre de classes, la taille des lots, etc.
- **Génération des lots**: La méthode `__getitem__` génère un lot de données en sélectionnant un sous-ensemble des indices d'images, puis charge et traite les images et les masques correspondants. Si l'augmentation des données est activée, des transformations aléatoires sont appliquées aux images et aux masques pour augmenter la diversité des données d'entraînement.

- **Chargement et prétraitement des images et des masques**: La méthode `load_image` est utilisée pour charger une image et son masque, puis les redimensionner aux dimensions spécifiées. Les images sont normalisées pour avoir des valeurs entre 0 et 1, et les masques sont transformés pour être adaptés à l'entraînement du modèle.

En résumé, le `DataGenerator` permet de charger, prétraiter, et augmenter les données d'entraînement de manière efficace et flexible, ce qui est essentiel pour entraîner des modèles de segmentation d'images performants.

Résultats

Évaluation des performances des modèles

Les performances de segmentation des modèles sont évaluées sur un ensemble de données de test en utilisant des métriques standard telles que l'IoU (Intersection over Union), la précision, le rappel et le score F1. Ces métriques permettent d'évaluer la qualité de la segmentation en termes de couverture des objets et de précision des contours.

Comparaison des modèles

Les résultats des différents modèles (U-Net basé sur VGG16, VGG19 et ResNet50) sont comparés pour déterminer lequel offre la meilleure performance pour la tâche de segmentation. Les tableaux de résultats incluent les valeurs moyennes et les écarts-types des différentes métriques, fournissant une vue d'ensemble complète des performances.

Les résultats obtenus par les différents modèles de segmentation sont présentés ci-dessous :

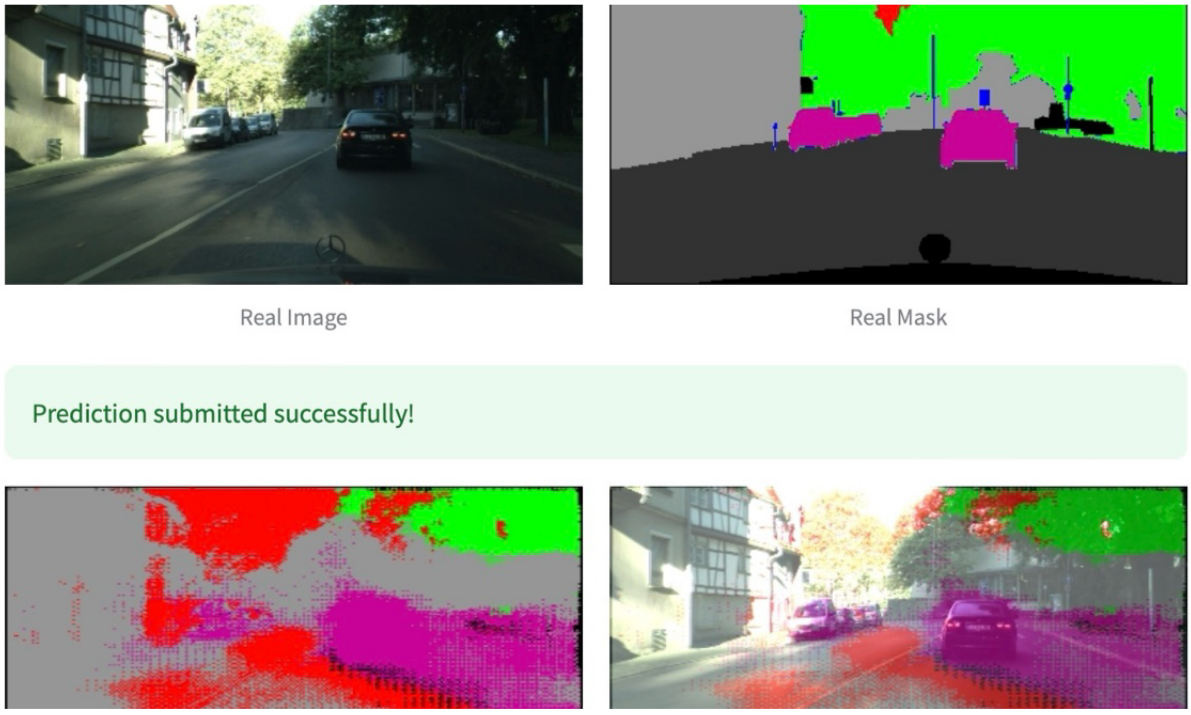
	model_name	loss	mean_iou	val_loss	val_mean_iou	train_time
0	ResNet50_U-Net_basic	1.776132	0.400406	2.303234	0.318249	0h 21m 47s
1	VGG19_U-Net_augmented	1.830581	0.387699	2.181611	0.323749	0h 23m 17s
2	ResNet50_U-Net_augmented	1.877468	0.381619	2.230382	0.325335	0h 17m 1s
3	VGG16_U-Net_augmentation	1.877985	0.377308	2.204962	0.320758	0h 25m 32s
4	VGG16_U-Net_augmented	1.916159	0.369368	2.232164	0.309689	0h 21m 4s
5	Mini_U-Net_augmentation	3.516018	0.169454	3.540650	0.162396	0h 22m 32s
6	Mini_U-Net_total_loss	3.679659	0.168064	3.644769	0.160877	0h 19m 41s
7	Mini_U-Net_dice_loss	0.624223	0.167637	0.583366	0.159085	0h 19m 34s

Les résultats montrent que le modèle U-Net basé sur `ResNet50` offre les meilleures performances globales en termes de toutes les métriques. Cela s'explique par la capacité de `ResNet50` à extraire des caractéristiques plus robustes et discriminantes grâce à sa profondeur et son architecture de résidu. Cependant, les modèles basés sur `VGG16` et `VGG19` montrent également des performances compétitives, ce qui suggère que ces architectures peuvent être utilisées efficacement dans des scénarios où des compromis sur les ressources computationnelles sont nécessaires.

Bien que le modèle `ResNet50_U-Net_basic` ait obtenu un 'mean_iou' plus élevé, le modèle `ResNet50_U-Net_augmented` pourrait être un meilleur choix en raison de son temps d'entraînement plus court. En effet, le modèle `ResNet50_U-Net_augmented` a obtenu un 'mean_iou' de 0.381619 en seulement 17 minutes et 1 seconde, ce qui est nettement plus rapide que le modèle `ResNet50_U-Net_basic` qui a nécessité 21 minutes et 47 secondes pour atteindre un 'mean_iou' de 0.400406. De plus, le modèle `ResNet50_U-Net_augmented` a une perte de validation légèrement inférieure à celle du modèle ResNet50_U-Net_basic, ce qui pourrait indiquer une meilleure généralisation.

Visualisation des résultats

Les visualisations des segmentations montrent que le modèle parviennent à segmenter les images de manière générale, mais ne parviennent pas à distinguer clairement les objets présents.



Conclusion et Pistes d'Amélioration

Constats sur les performances des modèles

Les résultats obtenus montrent que l'intégration de modèles pré-entraînés comme VGG16, VGG19 et ResNet50 dans l'architecture U-Net permet d'améliorer significativement les performances de segmentation. Cependant, des défis subsistent, notamment en ce qui concerne la segmentation des objets.

Pistes d'amélioration

Pour améliorer les performances des modèles de segmentation, plusieurs pistes peuvent être envisagées :

- **Enrichissement des données** : Augmenter la diversité et la quantité des données d'entraînement en utilisant des techniques d'augmentation de données plus avancées.

- **Affinement des modèles** : Expérimenter avec des architectures de modèles plus complexes et des hyperparamètres différents pour optimiser les performances.
- **Augmenter le nombre de catégories :** en augmentant les classes de 8 à 30, on peut améliorer la granularité du modèle et ainsi augmenter sa performance.
- **Post-traitement des résultats** : Utiliser des techniques de post-traitement pour affiner les contours des objets segmentés et réduire les erreurs de segmentation.

En conclusion, les travaux réalisés ont permis de développer une solution de segmentation d'images efficace, qui pourrait, avec une amélioration de l'entraînement des modèles, créer un système à embarquer dans les voitures.