******

***Véhicule Autonome***

Introduction :

Le véhicule autonome désigne un véhicule capable de se déplacer tout seul sans l’aide du conducteur. Le lancement de la Google Car en 2010 a donné une véritable impulsion à cette innovation, aujourd’hui au cœur des stratégies de développement des constructeurs automobiles.

Le véhicule autonome est un véhicule remplissant la fonction de transporter d’un point à un autre des biens et/ou des passagers sans intervention humaine. Il est muni d’un système de pilotage automatique lui permettant de se déplacer dans les conditions réelles de circulation. Actuellement en développement, il sera l’aboutissement, après de nombreuses années de recherche, du véhicule intelligent, qui remplace le conducteur humain.

Le véhicule autonome peut aussi gérer des systèmes de moindre importance, comme la climatisation. Mais là n’est pas l’enjeu principal de cette technologie. En effet, son intérêt est d’assister le conducteur dans sa conduite, partiellement ou totalement. Comme avec le freinage d’urgence, ou en prenant le contrôle d’une fonction critique, par exemple le régulateur de vitesse. Un véhicule totalement autonome possède la gestion totale des fonctions critiques, à savoir moteur, accélérateur, direction, freins

Enfin, le véhicule autonome doit donc être capable de se déplacer, de choisir le trajet vers sa destination, de se localiser précisément en temps réel, de conduire en respectant le code de la route, de détecter et analyser son environnement (obstacles, conditions climatiques, usagers de la route, signalisation, chaussée). Afin d’adapter son déplacement en position et en vitesse pour éviter les obstacles et les autres usagers de la route.

Différents types d’algorithme exécutent ces fonctions de localisation en temps réel de l’environnement. Nous analyserons les résultats de chacun d’eux ainsi que des méthodes permettant d’améliorer les résultats de segmentation d’images. Nous présenterons les résultats d’autres équipes de recherche sur ce sujet et enfin conclure par des pistes d’amélioration pour nos résultats.

1. Augmentation des données

Le rôle de l’augmentation des données est d’améliorer la qualité des prédictions de façon significative en augmentant artificiellement le volume de données à disposition pour l’entraînement du modèle.

L’augmentation des données consiste à réaliser des transformations aux données avant de les utiliser pour l’entraînement tout en conservant leur structure. Pour chaque observation, on crée plusieurs variations. Ainsi le volume de données est artificiellement multiplié.

Dans ce cas, l’augmentation des données peut consister en : des rotations, déformations, transposition, changements de couleurs, ajout de bruit ou autres. Pour d’autres types de données, on peut appliquer des fonctions mathématiques simples (log, puissances), combiner des caractéristiques pour en créer une nouvelle, créer des groupes, ou appliquer des techniques plus avancées comme la [PCA](https://fr.wikipedia.org/wiki/Analyse_en_composantes_principales) ou l’[auto-encoding](https://fr.wikipedia.org/wiki/Auto-encodeur).

La data augmentation permet d’introduire du chaos dans les aspects insignifiants des données pour empêcher le modèle de les utiliser pour tirer des conclusions. On le force ainsi à voir ce que l’humain verrait.

Il est alors impératif que toutes les variantes de nos images soient exclusivement dans le jeu de données d’entraînement, ou dans celles de test, mais pas les deux sinon nous observerons une fuite de données entre le jeu d’entrainement et celui de test.

1. Jeu de données

Notre étude se base sur l’utilisation du jeu de données Cityscapes. Celui-ci est composé de 2975 images qui nous servirons de jeu de données d’entrainement de nos différents modèles et un jeu de 500 images qui nous servira de jeu de validation de nos modèles.

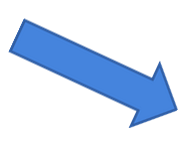
Nos 2 jeux sont bien distincts, c'est-à-dire qu’aucune image ne fait partie des 2 jeux. Les jeux d’entrainement et de validation sont bien indépendants. Il n’y a donc aucune fuite de donnée entre les 2 jeux d’image.

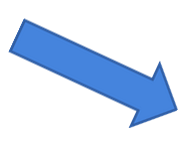
L’évaluation des différents modèles se fera donc grâce au jeu de validation. Nous comparerons les prédictions de segmentation obtenues et la segmentation théorique pour ces images composant le jeu de validation.

1. Score utilisé pour la segmentation d’image

Nous traitons ici d’une problématique de segmentation d’image. En effet, notre but est de catégoriser les différentes parties d’une image initiale. Nous souhaitons identifier les véhicules, les passants, la route etc…

Exemple d’image tirée du dataset Cityscapes et les catégories associées :

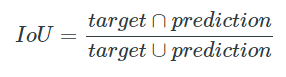
 



La segmentation d'image est une opération de [traitement d'images](https://fr.wikipedia.org/wiki/Traitement_d%27image) qui a pour but de rassembler des [pixels](https://fr.wikipedia.org/wiki/Pixel) entre eux suivant des critères pré-définis. Les pixels sont ainsi regroupés en régions, qui constituent un pavage ou une partition de l'image. Il peut s'agir par exemple de séparer les objets du fond.

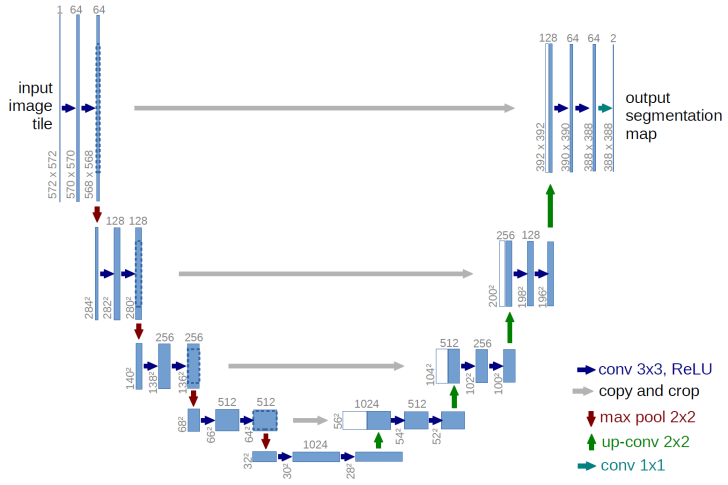
Il est important de bien choisir la mesure permettant de qualifier les performances de nos modèles car une mesure mal adaptée peut donner des résultats trompeurs. La tâche de segmentation sémantique est de prédire la classe associée à chaque pixel d’une image.

La métrique Intersection over Union (IoU), également appelée indice Jaccard, est essentiellement une méthode pour quantifier le pourcentage de chevauchement entre le masque cible et notre sortie de prédiction. Cette métrique est étroitement liée au coefficient de Dice qui est souvent utilisé comme fonction de perte pendant l’entraînement. Pour résumer, la métrique IoU mesure le nombre de pixels communs entre la cible et les masques de prédiction divisé par le nombre total de pixels présents sur les deux masques.



1. Modèle de référence : UNET

Ci-dessous l’architecture d’un modèle UNET qui nous servira de modèle de référence pour comparer aux résultats des différents modèles entrainés.



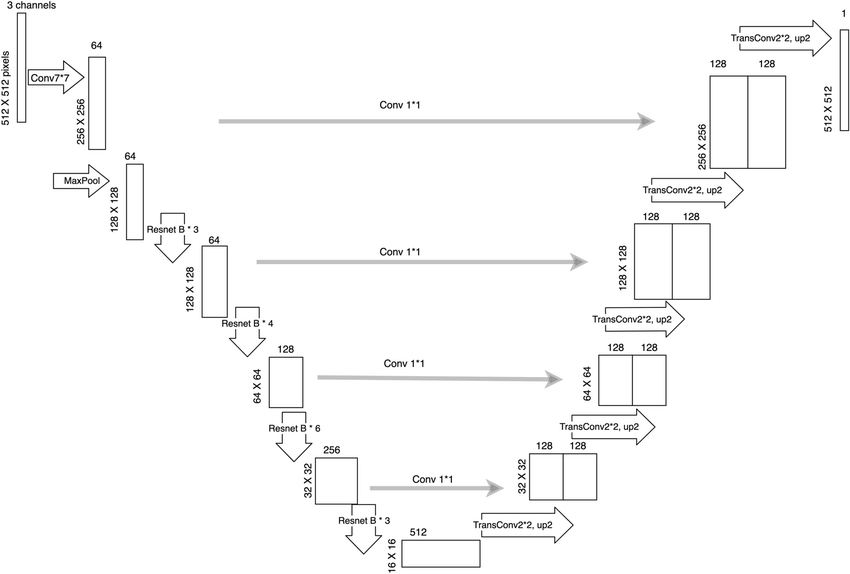
Le tableau ci-dessous reprend les résultats du modèle UNET. Ces résultats serviront du modèle de référence pour comparer les performances des différentes architectures décrites dans cette note. Les performances ont été obtenues avec augmentation de donnée lors de l’entrainement du modèle. Le score IoU obtenu est de **0.9947**

|  |  |  |
| --- | --- | --- |
| **avec augmentation** | | |
| **loss** | **iou\_score** | **f1\_score** |
| **0.8794** | **0.9302** | **0.9311** |

1. Méthode UNET et différents backbones

Nous avons défini une référence de comparaison pour comparer les performances de nouvelles architectures. Nous allons maintenant entrainer différents modèles basés sur une architecture UNET avec différents backbones :

* + ResNet
  + SE-ResNet



Voilà un exemple d’architecture UNET avec encodeur.

Nous retrouvons ci-dessous les performances obtenues grâce à ces différentes architectures. Performances obtenues en considérant toujours le jeu de validation :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **sans augmentation** | | |
|  |  | **loss** | **iou\_score** | **f1\_score** |
| **ResNet** | resnet34 | **0.1482** | **0.9557** | **0.9564** |
| **SE-ResNet** | seresnet34 | **0.0748** | **0.9699** | **0.9704** |

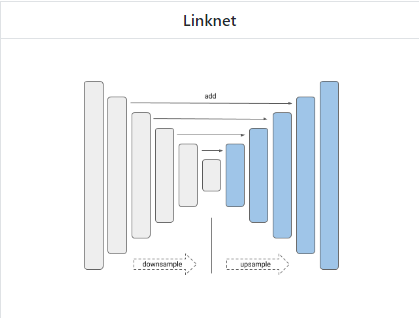
Nous observons déjà une amélioration des performances puisque nous passons de 0.9557 avec un minimum 0.9699 avec l’encodeur SE-ResNet .

Comme présenté précédemment, une méthode pour améliorer les performances d’un modèle s’opère en utilisant l’augmentation de donnée. Voyons les résultats dans les mêmes conditions en ajoutant simplement lors de la phase d’apprentissage l’augmentation de donnée du jeu d’entrainement.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **sans augmentation** | | | **avec augmentation** | | |
|  | **loss** | **iou\_score** | **f1\_score** | **loss** | **iou\_score** | **f1\_score** |
| resnet34 | **0.1482** | **0.9557** | **0.9564** | **0.0851** | **0.9805** | **0.9807** |
| seresnet34 | **0.0748** | **0.9699** | **0.9704** | **0.1303** | **0.9710** | **0.9714** |

Les performances obtenues sur le jeu de validation suite à l’entrainement du modèle avec augmentation de donné du jeu d’entrainement confirme l’amélioration de la performance grâce à cette technique. L’augmentation de donnée a donc amélioré les performances de tous les modèles.

1. Méthode Linknet et différents backbones

Nous allons maintenant tester les performances d’une autre architecture : Linknet. Cette architecture sera elle aussi testée successivement avec différents encodeurs ou backbones. Nous comparerons là-aussi les performances avec et sans augmentation de donnée lors de l’entrainement.

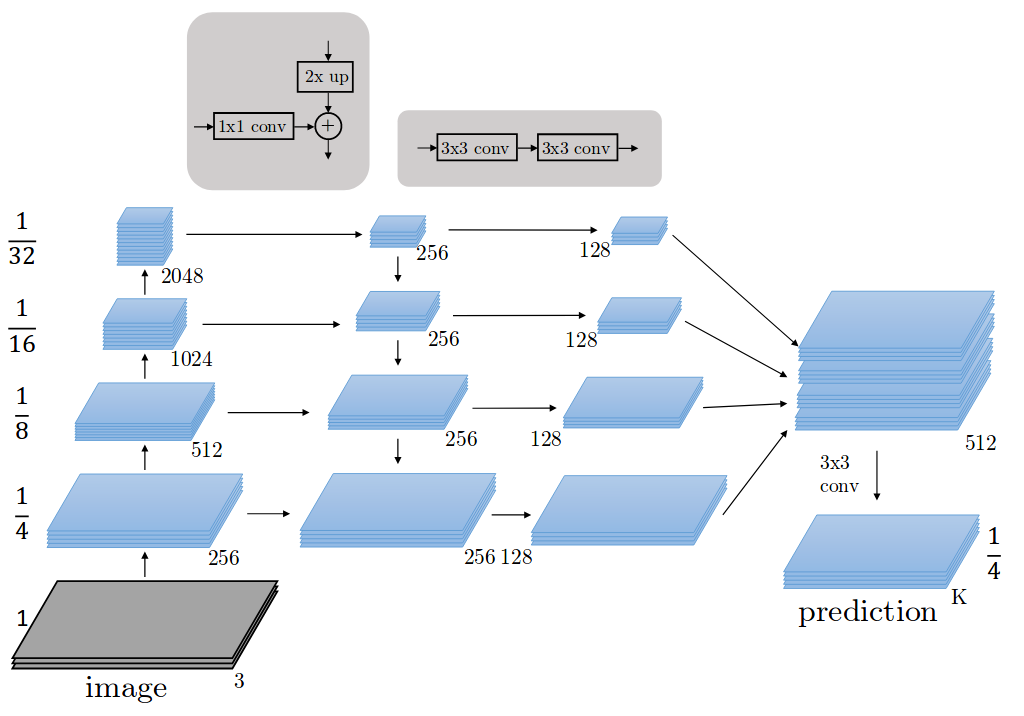
Les performances sont résumées dans le tableau ci-dessous :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **sans augmentation** | | | **avec augmentation** | | |
|  | **loss** | **iou\_score** | **f1\_score** | **loss** | **iou\_score** | **f1\_score** |
| resnet34 | 0.3539 | 0.5260 | 0.6463 | 0.3082 | 0.5753 | 0.6919 |
| densenet201 | 0.7918 | 0.1302 | 0.2088 |  |  |  |
| efficientnetb2 | 0.4140 | 0.4494 | 0.5866 |  |  |  |

Nous voyons là encore que l’augmentation de donnée permette d’améliorer les performances. Si nous comparons les 2 architectures, Unet et Linknet, la nouvelle architecture n’améliore pas les performances de segmentation. Les meilleurs résultats restent obtenus grâce au modèle Unet.

1. Méthode FPN et différents backbones

Nous allons maintenant tester les performances d’une autre architecture : FPN. Cet acronyme signifie Feature Pyramide Network. Cette architecture sera elle aussi testée successivement avec différents encodeurs ou backbones. Nous comparerons là-aussi les performances avec et sans augmentation de donnée lors de l’entrainement. Un exemple de cette architecture est présenté ci-joint :



Les performances de l’architecture FPN sont résumées dans le tableau ci-dessous :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **sans augmentation** | | | **avec augmentation** | | |
|  | **loss** | **iou\_score** | **f1\_score** | **loss** | **iou\_score** | **f1\_score** |
| resnet34 | 0.3283 | 0.5599 | 0.6719 | 0.3631 | 0.5106 | 0.6371 |
| efficientnetb2 | 0.6561 | 0.2408 | 0.3440 |  |  |  |

Dans ce cas d’architecture, l’augmentation de donnée n’a pas amélioré significativement les performances du modèle. Les performances avec le backbone de type resnet restent cependant, avec et sans augmentation de donnée, meilleures que les performances de la référence.

1. Conclusion

Pour conclure sur la comparaison des performances des différents modèles, nous observons une amélioration de la majorité des modèles grâce à l’utilisation de l’augmentation de donnée lors de l’entrainement.

Le graphe ci-dessous résume les meilleurs résultats des différentes architectures.

La meilleure performance a ainsi été obtenue par l’architecture **Unet avec un backbone de type Resnet**.

1. Améliorations envisagées

Dans un objectif d’amélioration des performances du modèle Unet/Resnet, 2 changements ont été appliqués pour évaluer l’impact sur la performance de classification.

Le 1er changement a été de changer de Loss fonction, passant ainsi d’une fonction Dice Loss utilsée pour les résultats présentés précédemment à une fonction loss de type cross-entropy.

Les résultats ci-dessous montrent qu’il n’y a pas d’amélioration en utilisant la fonction de cross-entropy.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **loss** | **iou\_score** | **f1\_score** |
| Dice Loss | **0.8772** | **0.9881** | **0.9888** |
| cross entropy | **0.0851** | **0.9805** | **0.9807** |

Exemple de masque prédit :

Une image contenant texte, afficher

Description générée automatiquement Une image contenant texte, moniteur, équipement électronique, afficher

Description générée automatiquement

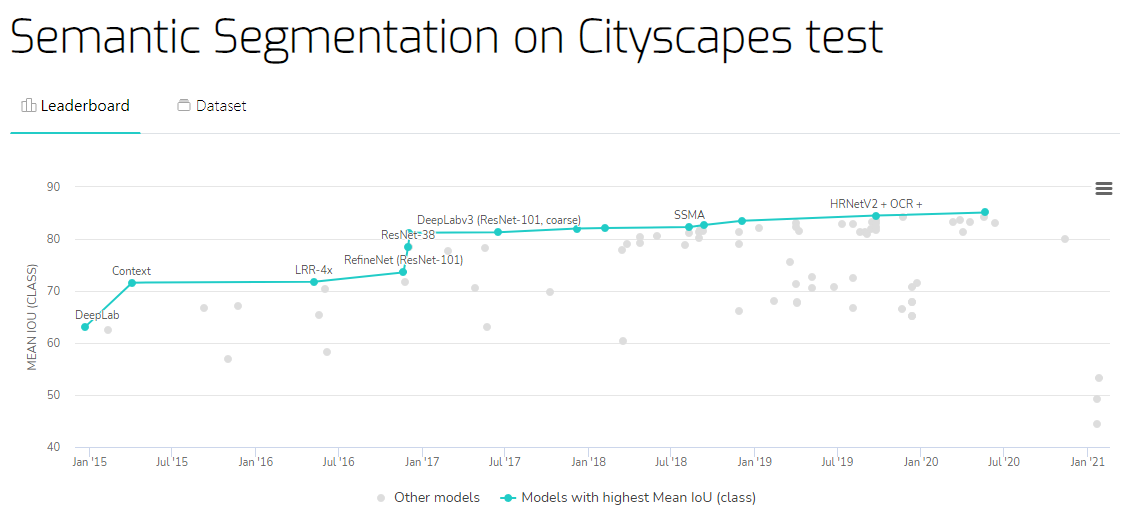
Le 2ième changement effectué est le temps d’entrainement. Les résultats sont légèrement améliorés mais le temps de traitement est très élevé.

Augmentation du nombre d’epoch pour un IoU score final de :

**1.0000** pour un temps d’entrainement de **6h53min**

**0.9955** pour un temps d’entrainement de **3h58min**

Enfin, en regardant les études et simulations réalisées avec ce dataset, les performances obtenues sont les suivantes :





Enfin, une dernière piste d’analyse serait de regarder le balancement des différentes classes. En effet, les catégories ne sont pas réparties de manière homogène. Il serait intéressant de calculer le score IoU pour chaque classe pour identifier les erreurs de segmentation et les améliorer plus efficacement.