***Note méthodologique***

**Une étude comparative entre U-Net Mini et SegFormer pour la segmentation sémantique**

**I. Introduction**

La segmentation sémantique occupe aujourd’hui une place très importante dans le domaine de la vision par ordinateur ou computer vision en anglais. Elle permet de classifier des images en attribuant une étiquette de classe à chaque pixel. Son objectif est de regrouper les pixels représentant des objets similaires et de leur accorder une catégorie prédéfinie. Elle trouve son application dans divers domaines dont la voiture autonome, la surveillance vidéo intelligente et les diagnostics médicaux précis. Le récent essor des réseaux de neurones convolutifs (CNN) a permis de réelles avancées dans le domaine de la segmentation sémantique d’images. Des modèles comme U-Net ou VGG16 ont permis d’atteindre des performances sans précédent. Mais les CNN, bien que performants, présentent plusieurs faiblesses dont la dépendance aux données massives, leur incapacité à capturer le contexte global et un besoin important en termes de puissance de calcul. C’est dans ce cadre que les Transformers ont fait leur entrée dans ce domaine. En effet, leur capacité de modélisation des dépendances de longue portée et leur légèreté relativement aux CNN ont fait d’eux un outil de choix dans les tâches de segmentation sémantique.

Dans ce papier, nous proposons une étude comparative entre un modèle U-Net Mini qui nous sert de baseline et un SegFormer qui est considéré comme le state of the art des modèles de segmentation sémantique. Le SegFormer est un modèle qui s’appuie sur l’architecture Transformers.

Pour les besoins de cette étude, nous utilisons les modèles pré entrainés « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » pré entrainé sur les données Cityscapes et le *«****Nvidia/segformer-b0-finetuned-ade-512-512***» pré entrainé sur les données ADE20K.

**II. Dataset retenu**

Les données utilisées dans ce projet sont fournies par CityScapes. Nous avons un jeu de données de 5000 images d’une caméra embarquée. Ces images sont fournies avec leurs masques annotés des différentes catégories.

En tout, nos images sont annotées de 8 catégories principales et de 32 sous catégories.

Pour les besoins du projet nous n’aurons besoin que des 8 catégories principales qui sont : vide, route, construction, objets, nature, ciel, humain et véhicule. Par conséquent, nous opérant une petite manipulation qui vise à convertir nos masques à 34 canaux en des masques à 8 canaux uniquement où chaque canal représente une catégorie. Cette manipulation retourne, ensuite, l’indice du canal avec la valeur maximale pour chaque pixel, ceci donne la catégorie principale pour chaque pixel. 4

Le jeu de données comprend deux dossiers, un dossier contenant les images brutes provenant d’une caméra embarquée et un autre contenant les masques de la segmentation sémantique.

Chaque dossier contient 3 sous dossiers qui sont : test, train et val qui ne sont autres que les données de test, d’entrainement et de validation.

Le dossier « leftImg8bit » contient des images en couleur RGB en format « png » de dimension 2048 x 1024. Ce fichier comprend 5000 images

Le dossier « gtFine » contient, quant à lui, les masques. En tout à chaque image correspond 4 fichiers de masques :

- Un fichier « \_gtFine\_color.png » au format « png ». Cette image représente une vue segmentée d’une scène de rue. Chaque pixel a une couleur RGB qui correspond à une classe spécifique. Ces couleurs sont utilisées pour des tâches de segmentation sémantique. Cela permet de visualiser facilement les différentes classes dans l’image.

- Un fichier Json nommé « \_getFine\_polygons.json », ce fichier contient les annotations polygonales pour chaque objet dans l’image. Les polygones définissent les contours des objets et sont utilisés pour générer les masques de segmentation. Le fichier Json est structuré de la manière suivante : ‘ImageHeight’ et ‘ImageWidth’ définissent la dimension de l’image, ‘objects’ constitue la liste des objets annotés, cette liste se compose d’un ‘label’ qui définis la classe de l’objet et de ‘polygon’ qui représente la liste des coordonnées x et y des contours des objets.

- Un fichier « \_gtfine\_labeIds.png » où chaque pixel a une valeur qui correspond à l’Id d’une classe spécifique. Ce fichier est très similaire de « \_gtFine\_color.png » mais est représenté en nuance de gris. Chaque pixel a une valeur allant de -1 à 32 représentants l’Id de la classe.

- Un fichier « \_getFine\_instanceIds.png » contenant des images en niveau de gris où chaque valeur de pixel représente une instance unique. Ce genre de fichier est utilisé dans des tâches de segmentation d’instances ce qui n’est pas l’objet de notre projet puisque nous réalisons une segmentation sémantique.

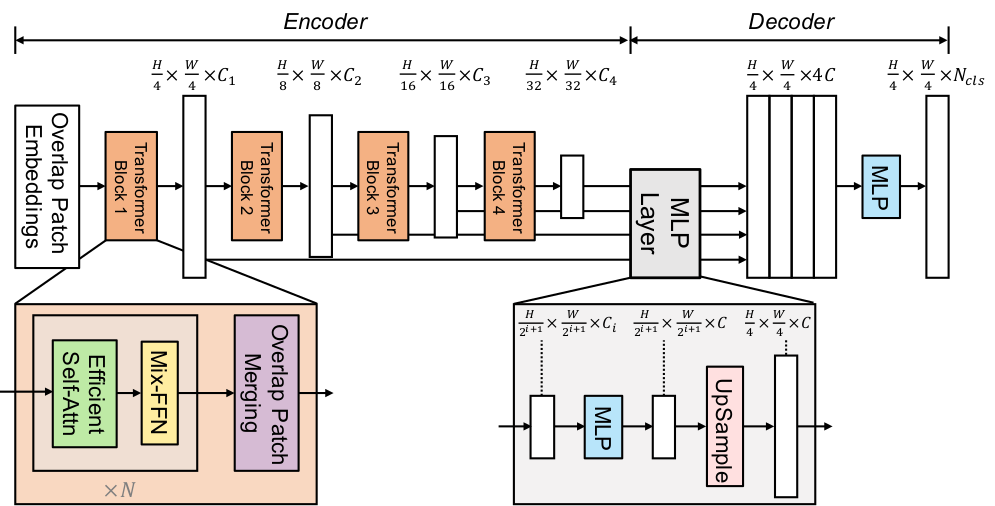
**III. Concept de l’algorithme SegFormer**

Le modèle SegFormer a été introduit en 2021 dans l’article « SegFormer : Simple and Efficient Design for Semantic Segmentation with Transformers » par Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez et Ping Luo.

Le modèle SegFormer combine une série d’innovations architecturales pour améliorer la segmentation sémantique. Le modèle se compose d’un encodeur basé sur une architecture Transformers hiérarchique et d’un décodeur MLP (Multilayers Perception) simple er léger. L’encodeur Transformer à structure pyramidale permet de traiter des caractéristiques à différentes échelles ce qui lui permet de détecter des détails fins mais également des contextes plus larges sans avoir recours au codage positionnel ce qui évite une baisse de performances due à une différence de résolution entre les données d’entrainement et de test. SegFormer intègre un simple décodeur MLP permettant de capturer, à la fois, l’attention locale et l’attention globale car il fusionne efficacement les informations des différentes couches de l’encodeur.

Cette architecture permet de simplifier le processus de décodage offrant une meilleure efficience en termes de performances et d’économie de ressources.

Pour conclure sur cette partie, et sans entrer dans des détails mathématiques fastidieux, nous pouvons dire que le modèle SegFormer et ses différentes, en particulier, ceux développés par Nvidia et disponibles dans la bibliothèque Transformers de Hugging Face, sont, du moins en théorie, plus efficients car à la fois plus performants et plus économes que les réseaux de neurones convolutifs. Cette efficacité vient de la combinaison d’un encodeur se basant sur l’architecture Transformers et d’un décodeur MLP léger.



**IV. Modélisation**

Dans ce papier, nous allons nous intéresser à deux variantes du modèle SegFormer : « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » pré entrainé sur les données Cityscapes et le *«****Nvidia/segformer-b0-finetuned-ade-512-512***» pré entrainé sur les données ADE20K. Les deux variantes ont été utilisées par les auteurs de l’article : « SegFormer : Simple and Efficient Design for Semantic Segmentation with Transformers ». Le B0 fait référence à la plus petite variante du modèle SegFormer, celui-ci utilise une famille d’encodeurs Transformers hiérarchiques appelés MiT-B0 pour Mix Transformer de taille B0. Malgré sa petite taille (comparé à SegFormer B5 par exemple), le B0 concède des performances très respectables. Nous utilisons SegFormer B0 car il offre le meilleur compromis entre performances et rapidité.

Le « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » est un SegFormer B0 pré entrainé dans un premier temps sur le jeu de données ImageNet puis fine tuned sur les données Cityscapes pour la tâche de segmentation sémantique d’images urbaines.

Le *«****Nvidia/segformer-b0-finetuned-ade-512-512***» suit la même logique mais a été fine tuned sur le jeu de données ADE20K.

Pour résumer donc, nous effectuons 3 entrainements, 2 avec le « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » avec ajustement de quelques paramètres et hyperparamètres et 1 avec le *«****Nvidia/segformer-b0-finetuned-ade-512-512***».

Leurs performances seront comparées à celle d’un U-Net Mini entrainé sans augmentation de données et produisant des performances satisfaisantes.

La comparaison portera sur plusieurs critères, d’abord la métrique « MeanIoU », ensuite la fonction de perte sur les données de validation « Val Loss », à noter que la fonction de perte utilisée est la « cross-entropy ». Le troisième point de comparaison est le temps de calcul et enfin, nous réaliserons des inférences pour juger visuellement de la qualité de segmentation de chacun des modèles entrainés.

Avant d’entrainer nos modèles, nous préparons 2 data generator : un premier pour charger et prétraiter les données que nous injectons dans notre modèle U-Net Mini et la version 2 du SegFormer « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** ». Le deuxième concerne la version 1 de notre SegFormer « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** ».

L’idée est d’entrainer le SegFormer « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » en lui injectant, tant tôt, des données avec une haute définition (Version 1 à 1024x1024) et tant tôt des données de plus faible définition (Version 2 à 512x512).

A noter que notre modèle baseline a été entrainé sans augmentation de données. Il nous paraissait, donc, évident d’entrainer nos modèle SegFormer sur des données non augmentées également. Le but étant de rendre la comparaison la plus fiable possible.

Il est, également, à noter qu’il existe une méthode permettant d’initialiser un processeur d’image (SegformerImageProcessor) à partir d’un modèle SegFormer pré entrainé.

Nous avons, tout d’abord, cherché à l’utiliser pour le pré traitement de nos données mais nous avons dû abandonner l’idée car les tenseurs résultants n’avaient pas les bonnes dimensions. Nous avons dû, de ce fait, les préparer manuellement, en quelques sortes, en ajoutant quelques lignes de code à notre data generator.

A noter que le U-Net Mini a été entrainé sur 20 époques contre 5 pour SegFormer avec ses deux variantes.

**V. Synthèse des résultats**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| | **Modèle** | | --- | | |  | | --- | |  |  |  | | --- | | **mIoU (%)** | | **Temps de calcul global (heures)** | **Validation Loss** |
| **U-Net Mini (20 époques)** | 69.38 | 3h30 | 0.3423 |
| **SegFormer-B0 (ADE20K, 512x512) (5 époques)** | 66.21 | 0h53 | 0.3469 |
| **SegFormer-B0 (Cityscapes, 1024x1024) v1 (5 époques)** | 76.39 | 3h18 | 0.2454 |
| **SegFormer-B0 (Cityscapes, 1024x1024) v2 (5 époques)** | 70.55 | 0h53 | 0.3036 |

L’examen du tableau ci-dessus montre que les modèles SegFormer et notamment la variante « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » produisent de meilleurs résultats et notamment sur la version 1. La mean IoU atteint 76.39% contre 69.38% pour le modèle U-Net Mini. La val loss ne déroge pas à cette règle, elle est de 0.2454 pour SegFormer et de 0.3423 pour le U-Net Mini. En termes de temps de calcul U-Net Mini met 3 heures et 30 minutes contre 3 heure et 18 minutes pour SegFormer avec 15 époques de moins.

C’est donc naturellement que le modèle SegFormer « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » dans sa version 1, c’est-à-dire, celle entrainée avec des images et masques de dimension 1024x1024, qui est déployé.





L’examen visuel confirme le constat empirique. SegFormer est nettement plus performant que le U-Net Mini et ce sur tous les plans.

**VI. Limites et améliorations possibles**

SegFormer présente plusieurs avantages par rapport à U-Net Mini. En effet, il produit de meilleures performances et le temps de calcul est nettement réduit.

Néanmoins, lorsqu’on réalise des inférences sur nos données de test, on s’aperçoit que la qualité graphique des masques laisse à désirer. Cela pourrait bien provenir de la taille des masques prédits car ceux -ci ont une dimension se sortie qui représente 1/4 de la résolution de l’image d’entrée originale (après son pré traitement) ce qui signifie que pour la variante *«****Nvidia/segformer-b0-finetuned-ade-512-512***», le masque de sortie aura une résolution de 128x128. Ceci a pour conséquence un masque pixélisé et qui manque de netteté lors des inférences.

Pour résoudre ce problème, nous avons essayé de redimensionner les masques prédit mais notre niveau de connaissances dans cette problématique n’a pas permis d’obtenir des résultats probants.

L’entrainement d’une variante plus sophistiquée comme la B5 pourrait, peut-être, résoudre cette problématique.

Autre limite, et non des moindres, c’est que bien que SegFormer soit plus rapide, certaines variantes demandent des ressources importantes pour fonctionner.

Alors que le *«****Nvidia/segformer-b0-finetuned-ade-512-512***», n’a aucun problème à tourner avec le Tesla L4 disponible sur Colab, le « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** », en revanche, ne peut tourner que sur le Tesla A100, autrement le calcul s’interrompt par manque de mémoire disponible. Le problème est que le Tesla A100 est rarement disponible ce qui n’est pas commode lorsque l’on est tenu par les délais.

La solution est de s’équiper du matériel adéquat avec toutes les contraintes que ceci peut poser.

Une autre amélioration que l’on pourrait apporter à l’avenir, c’est d’entrainer le modèle sur plus d’époques. Mon SegFormer « ***Nvidia/segformer-b0-finetuned-cityscapes-1024-1024*** » arrive à obtenir une mean-IoU de 76.39% avec seulement 5 époques et nous pensons qu’il est possible d’obtenir de meilleures performances en augmentant le nombre d’époques.