Stereolabs Sky Segmentation

Enoncé :

Développer un réseau qui segmente le ciel (style panoptic segmentation) sur des images couleur; l'implémentation devrait comprendre le training complet ainsi que l'inférence qui devra être exécutée en temps réel sur GPU sur des vidéos incluant l'affichage. Idéalement le code d'inférence sera séparé du code de training en utilisant par exemple un export en ONNX. Une implémentation d'inférence en C++ avec TensorRT sera fortement apprécié. Aucune pénalités sur l'utilisation de code existant, mais les choix devront être justifiés, l'objectif est le système complet.

La structure du code, la documentation expliquant les choix techniques, les performances de l'inférence en runtime ainsi que l'accuracy seront considéré d'importance égal pour l'évaluation

Dataset :

Pour ce projet le dataset utilisé est Skyfinder : https://zenodo.org/record/5884485. Skyfinder est un dataset spécialisé pour la segmentation du ciel par rapport au paysage. Les images sont de tailles variables mais relativement petite. Malgré le manque de variété dans le dataset (53 caméras fixes avec des milliers de photos chacune) ce dernier est adapté pour un petit projet de ce genre. Seules 100 images par caméra on été prises pour créer un dataset de taille raisonnable, et 5 caméra utilisées pour le set de validation. Cependant, le dataset n'est pas parfait, ce qui veut dire qu'un tri sur les images a été effectué en partie à cause d'images complètement noires ou corrompues.

Training :

Le training est réalisé en Python en exécutant le main dans le fichier train.py. Un modèle type U-net a été codé en utilisant le framework Pytorch. U-net est un réseau spécialisé dans la segmentation d'image avec une architecture relativement simple ce qui est idéal pour ce projet, pour réaliser le modèle je me suis basé sur l'article scientifique d'U-net : https://arxiv.org/abs/1505.04597. Même si U-net est principalement conçu pour de la segmentation médicale, la segmentation du ciel nécessite de définir une limite entre le sol, les décors, le relief et le ciel ce qui nécessite une délimitation précise surtout dans le cas de photo avec des étendues d'eau par exemple. De plus U-net est adapté pour des images relativement petites ce qui va être utilisé lors de ce projet afin d'éviter un processus d'entrainement trop long. Lors de l'entrainement, plusieurs augmentations ont été utilisés grâce à la librairie « albumentations » (Rotation ± 20°, HorizontalFlip, VerticalFLip (rare)). Les données ont aussi été normalisées avec une moyenne de 0 et une déviation standard de 1 afin d'augmenter les performances du modèle en les standardisant. L'entrainement ne s'est effectué que sur 5 époques à cause de la limite de temps. La précision finale de

l'entrainement est de 93.44% de pixels corrects, mais dans le cadre de la segmentation avec Unet un autre indicateur de performance plus intéressant est aussi utilisé : le « dice similarity coefficient » (DSC). Le DSC utilise la formule suivante :

$$DSC = \frac{2 * | A \cap B |}{| A | + | B |}$$

Avec A l'image prédite et B la ground thruth, $| A \cap B |$ le nombre de pixels activés communs entre les deux images et $| A | + | B |$ la somme de tous les pixels activés des deux images. Le DSC du modèle est de 0.86 ce qui est plutôt bon malgré une marge d'amélioration possible. En augmentant les epochs (test avec 10) les performances semblent augmenter vis-à-vis des indicateurs (93.5% accuracy et 0.86 DSC) mais à cause de la petite quantité de données le modèle commençait tout simplement à montrer des signes d'overfitting ce qui rendait le modèle moins performant sur des données inconnues.

Inference :

Initialement, il était prévu de faire réaliser l'inférence avec Tensor RT, c'est d'ailleurs pour cette raison qu'une fonction afin d'exporter le modèle en format ONNX a été codée. N'ayant aucune connaissance préalable de Tensor RT il fallait tout reprendre de 0 avec différents tutoriels et l'installation de toutes les librairies nécessaires. Après avoir commencé ce processus il ne semblait pas possible de finir l'inférence durant le temps imparti.

C'est donc ainsi que dans la continuité de l'entrainement, l'inférence s'est faite en Python aussi. Le pipeline commence par la lecture d'une vidéo qui est basé sur une fonction réalisée lors d'un projet précédent qui permet de moduler le FPS souhaité. À chaque image de la vidéo, l'image est mise à la bonne taille si nécessaire, le modèle estime le masque et Opencv permet d'afficher la vidéo originale à côté de la vidéo avec le masque ajouté en bleu semi-transparent.

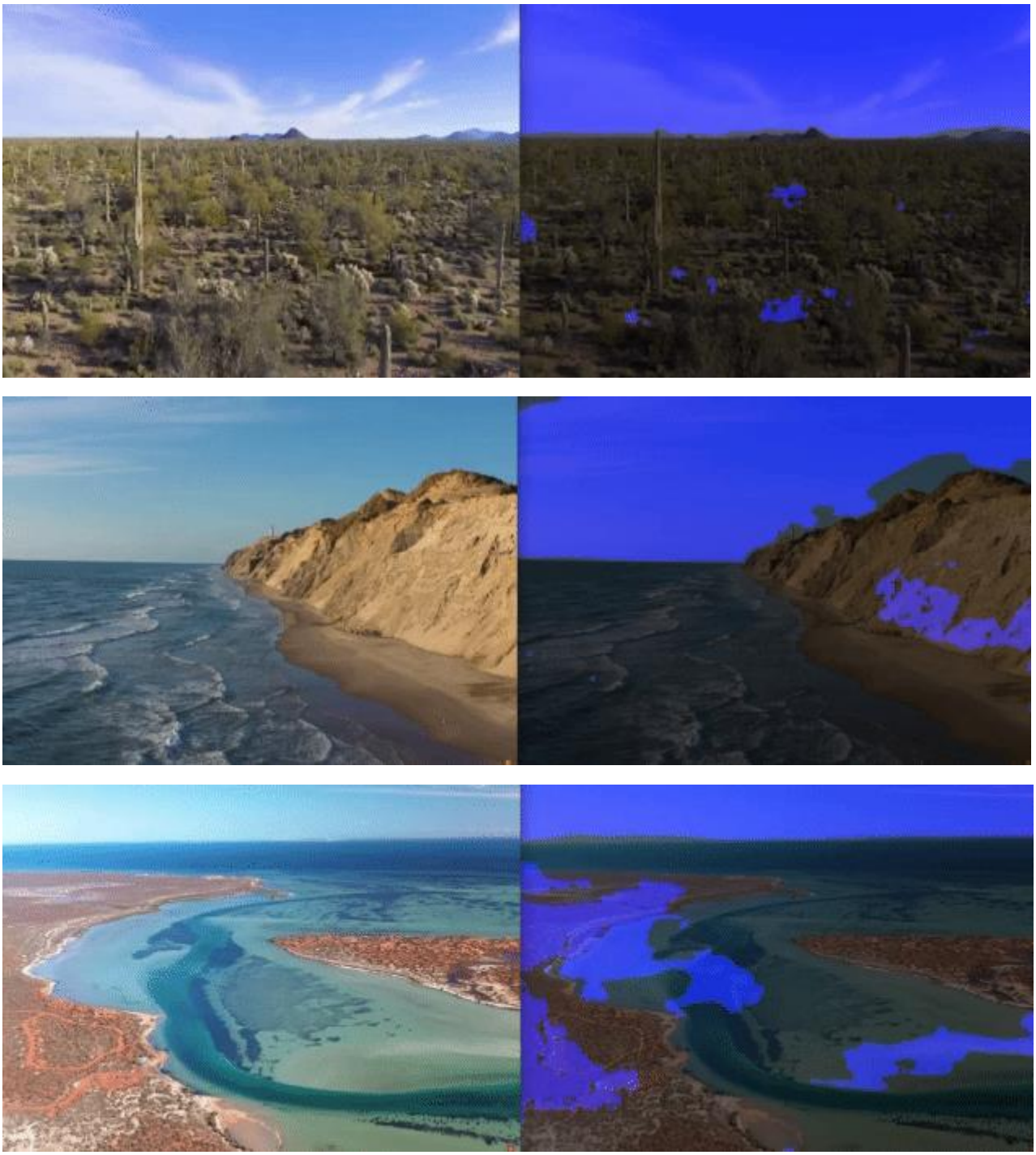
Les résultats sont les suivants (valeurs effectuées sur 2408 images de résolution 640 x 480 sur GPU NVIDIA 3060 Mobile):

Inférence seule: 13.57 imgs/seconde

Inférence avec affichage : 12.13 imgs/seconde

Inférence avec resize et affichage : 11.28 imgs/seconde

Le modèle n'est pas aussi rapide que si Tensor RT avait été utilisé mais il reste capable de tourner à 10 FPS de façon consistante. En regardant les gifs (dans le dossier, les pdf ne pouvant montrer de gif) ci-dessous on peut voir que le modèle fonctionne bien malgré l'apparition de bruit à certains endroits. On remarque que les points les plus difficiles sont la délimitation précise du ciel ainsi que la présence d'étendu d'eau qui perturbe le modèle. N'ayant pas particulièrement d'eau présente sur le dataset original, il n'est pas étonnant que le modèle ne soit pas très performant sur ce genre d'images



Images issues de la video suivante : https://www.youtube.com/watch?v=uu_B4ywAhOM