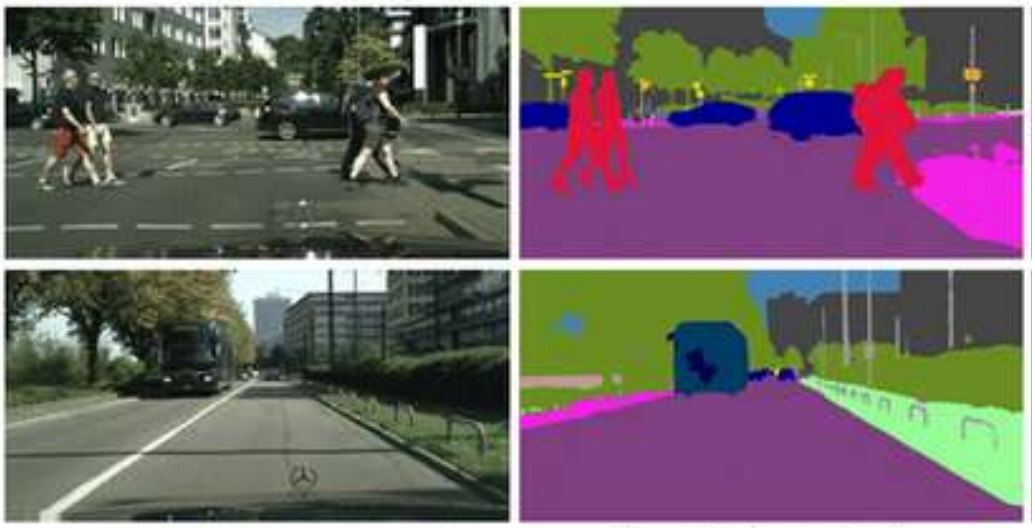
OpenClassrooms

- PROJET 8 -
Segmentation d’images

Note technique



Yann Héreng

https://github.com/yhereng/OpenClassrooms/tree/master/OC-P7-Image%20Segmentation

Table des matières

1. Introduction

1.1 Contexte

Le but du projet est de concevoir un premier modèle de segmentation d'images qui devra s'intégrer facilement dans la chaîne complète du système embarqué.

1.2 Cahier des charges

On souhaite une API qui prend en entrée l'identifiant d'une image et renvoie la segmentation de l'image de l'algorithme, et de l'image réelle. Le volume des données d'entraînement pouvant être important, on souhaite pourvoir manipuler ces données avec un objet type générateur permettant de fonctionner par batchs.

2. Jeu de données

Le jeu de données utilisé pour l'entraînement, la validation et les tests est issu du Cityscapes Dataset (https://www.cityscapes-dataset.com/dataset-overview/). Le jeu d'entraînement est constitué de 2975 images avec leurs masques respectifs, le jeu de validation comporte 500 images et masques. Le jeu de test est composé de 1200 images.

2.1 Génération des masques

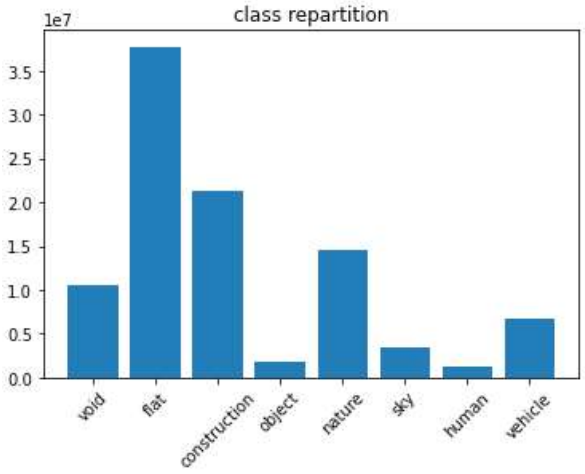
Le masque étant initialement une image en nuance de gris sur 30 catégories, une fonction à été implémentée pour permettre de grouper les catégories initiales en 8 catégories plus génériques et de générer une image plus parlante à l'affichage.

Group	Classes
flat	road · sidewalk · parking+ · rail track+
human	person* · rider*
vehicle	car* · truck* · bus* · on rails* · motorcycle* · bicycle* · caravan*+ · trailer*+
construction	building · wall · fence · guard rail+ · bridge+ · tunnel+
object	pole · pole group+ · traffic sign · traffic light
nature	vegetation · terrain
sky	sky
void	ground+ · dynamic+ · static+

Groupement des classes

Une image et son masque après pré-traitement

2.2 Répartition des classes



Répartition des classes de segmentation

On observe donc un répartition non homogène des classes. La métrique d'évaluation doit prendre en compte cette disparité. On notera également que certaines classes sont plus critiques que d'autres en terme de sécurité pour l'application finale.

3. Générateur

3.1 Principe

Afin de pouvoir manipuler un grand jeu de données ne pouvant être chargé en mémoire vive entièrement, une classe instanciant un objet de type générateur à été implémentée. Cet objet générateur permet de séparer le jeu de données en batchs. Les augmentations de données ainsi que le redimensionnement des images ont été intégrés à ce générateur pour rendre le pré-traitement automatique. Les framework Tensorflow et Keras sont compatibles avec ce type d'objet rendant alors la manipulation des données aisée.

3.2 Augmentation

Plusieurs techniques d'augmentation de données ont été implémentés dans le but d'étendre le jeu d'entraînement. Dans notre cas, 4 types d'augmentations ont été implémentés et testés: augmentation par ajout de bruit gaussien, augmentation par agrandissement aléatoire, augmentation par luminosité aléatoire et augmentation par inversement horizontal. Le but de l'augmentation est d'améliorer les performances de généralisation du modèle en l'exposant à une plus grande variété d'exemples d'entraînement. De manière évidente, les masques doivent subir le même pré-traitement que les images correspondantes.

image originale image floutée

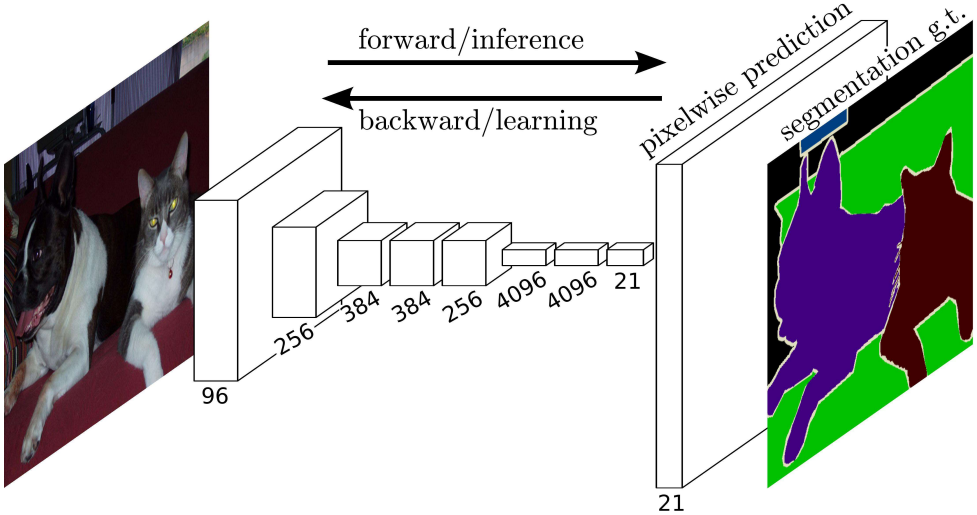
image agrandie image surexposée image inversée

4. Architecture des réseaux de neurones

Dans notre étude, nous avons mis en place et entraîné deux architectures de réseaux de neurones.
Le principe général étant d'utiliser uniquement des couches de convolution donnant leur nom à ce type de réseau neuronaux : Fully Convolutional Networks.
Les architectures s'articulent autour d'un encodeur et d'un décodeur. L'encodeur apprend les caractéristiques de haut niveau de l'image, les couches de convolution combinées aux couches de sous-échantillonnage fournissent en sortie un tenseur de basse résolution. Le décodeur prend en entrée ce tenseur et le sur-échantillonne en une image de sortie.

1.3 VGG-16 et FCN-8

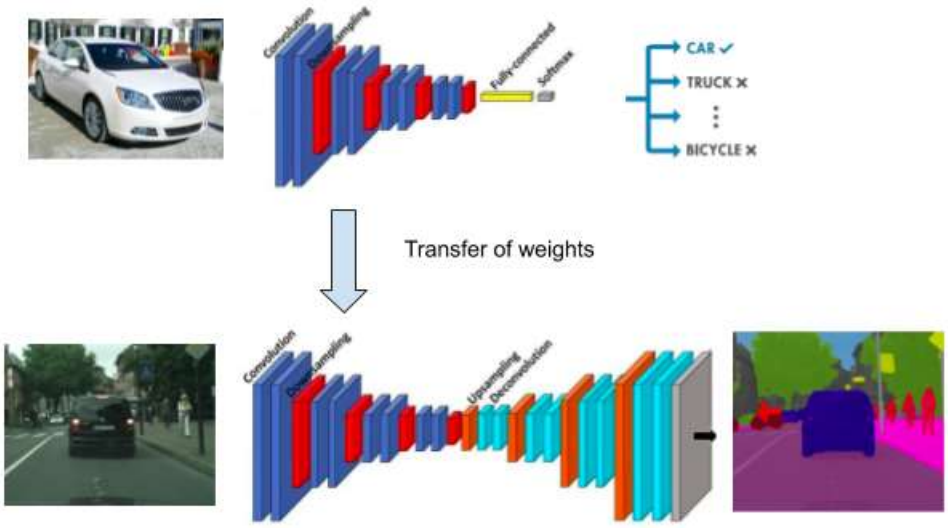
La première architecture est simple, elle consiste en un encodeur et un décodeur comme décrit ci-dessous.



Architecture d'un réseau type FCN

La partie encodeur est un VGG-16 sans les dernières couches fully connected. Cet encodeur est suivi d'un décodeur de type FCN-8 qui concatène les sorties du VGG16 depuis les stade 5,4 et 3.

Dans notre cas, nous avons utilisé la technique du tranfert learning du VGG-16, nous avons utilisé les poids d'un réseau pré-entrainé.

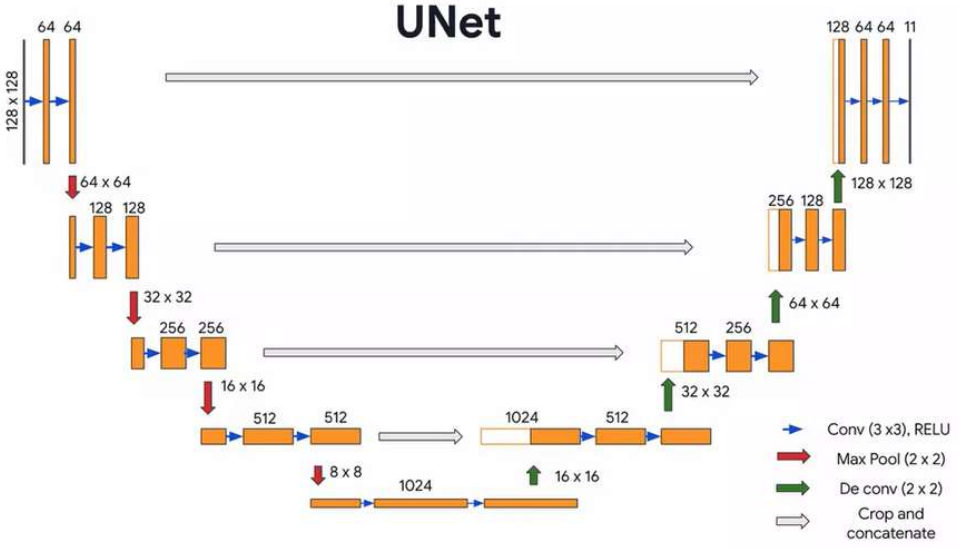


Principe du transfert learning

1.4 UNet

La première architecture présente l'inconvénient de perdre en résolution car travaillant sur les couches basses de l'encodeur. Pour pallier à cela, une deuxième architecture à été implémentée et testée.

Cette deuxième architecture est de type Unet, mettant en œuvre le principe de saut de connexions. Cette technique consiste à alimenter directement les couches de niveau élevé du décodeur par la sortie de la couche correspondante de l'encodeur. Les principaux intérêts sont d'éviter le phénomène de gradient évanescent et de favoriser la convergence du réseau.



Architecture UNet

1.5 Métrique

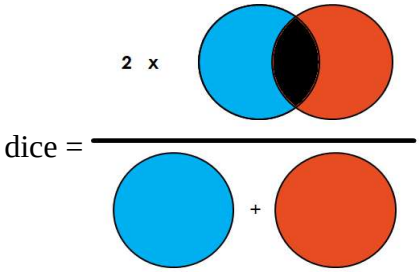
Afin de prendre en considération le déséquilibre des classes, nous avons utilisé les métriques Dice et IoU similaires à une métrique de type F1 score. Ces quantités sont calculées suivant les formules suivantes :

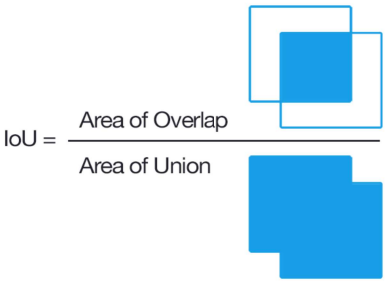
$$\mathrm{DC} = \frac{2TP}{2TP + FP + FN} = \frac{2|X \cap Y|}{|X| + |Y|}$$

$$\mathrm{IoU} = \frac{TP}{TP + FP + FN} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

Tp : True posisitve | FP : False Positive | FN : False Negative

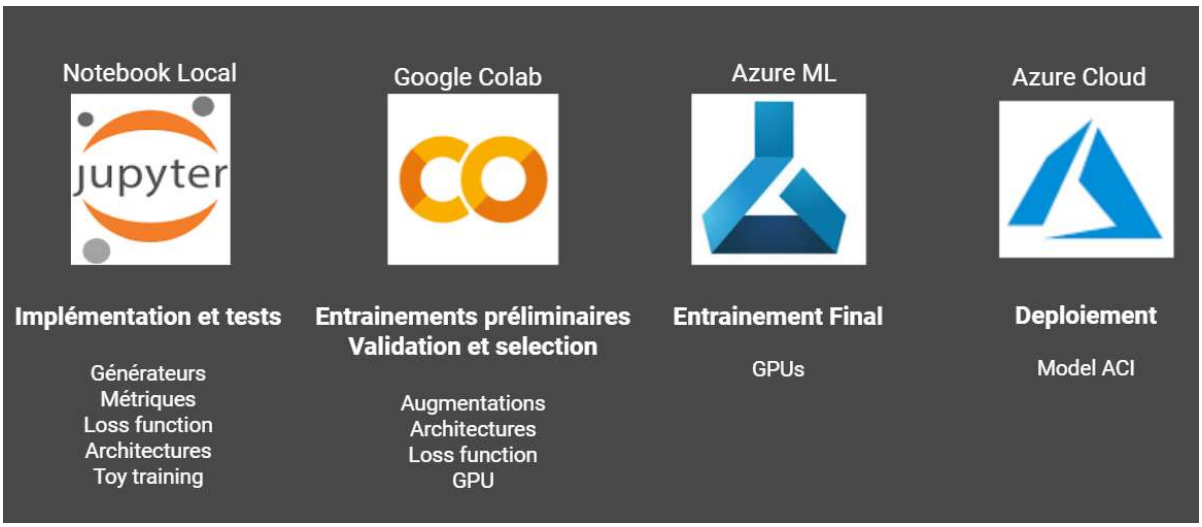
intuitivement, dans les deux cas, le dénominateur permet de ramener la mesure au niveau global comme illustré ci-dessous.





5. Méthodologie

Afin d'assurer une avancée continue du projet, dans le respect des contraintes de temps et de ressources, le schéma de travail suivant a été suivi.



Les différentes étapes suivies et leurs outils

6. Résultats préliminaires

Après avoir entraîné les différentes architectures, augmentation, et hyper-paramètres, nous obtenons les résultats suivants :

model_type	aug_type	loss_fcn	train_dice	val_dice	train_IoU	val_IoU	train_accuracy	val_accuracy	training_time (s)
fcn8	Blur & flip	categorical crossentropy	0,815	0,798	0,688	0,665	0,873	0,852	4345
fcn8	none	categorical crossentropy	0,803	0,774	0,671	0,632	0,866	0,834	2216
unet	Blur & flip	categorical crossentropy	0,741	0,729	0,590	0,575	0,822	0,818	4098
unet	none	categorical crossentropy	0,742	0,699	0,590	0,538	0,824	0,770	1992
unet	none	dice loss	0,636	0,595	0,467	0,424	0,636	0,594	1995
fcn8	none	dice loss	0,388	0,381	0,241	0,236	0,388	0,381	2229

Comme attendu, l'augmentation permet d'améliorer significativement le score du modèle, cependant,les conséquences en terme de temps d'entraînement ne nous à pas semblé justifier l'opération. Les résultats nous ont logiquement conduit à continuer le travail en sélectionnant l'architecture FCN-8, sans augmentation et une fonction perte 'categorical cross entropy'

7. Entraînement final

7.1 Early stopping

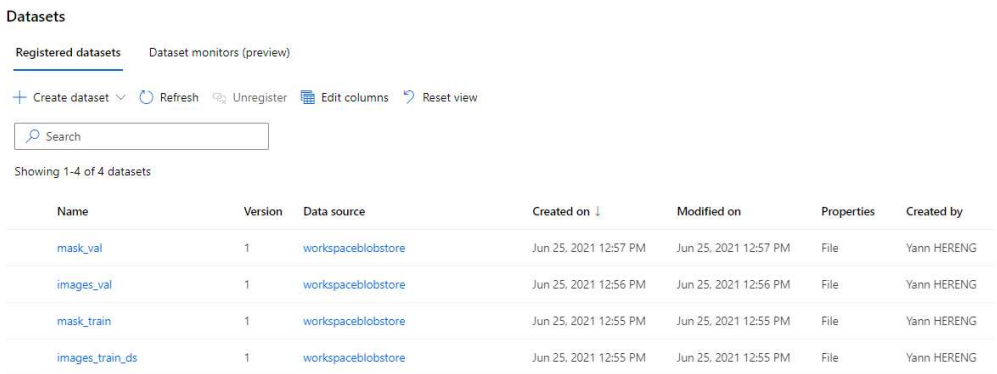
Afin d'éviter le sur-entraînement et d'optimiser la consommation des ressources, nous avons mis en place un callback de type earling stopping avec une patience de 6 époques sur la métrique IoU du jeu de données de validation. L'entraînement s'arrête si, pendant 6 époques, l'IoU ne s'est pas amélioré.

7.2 ModelCheckPoint

Afin de s'assurer de sauvegarder les poids d'entraînement du meilleur modèle, nous avons mis en place un callback de type Model checkpoint. En effet, les poids résultant de la dernière époque d'entraînement ne sont pas nécessairement les poids donnant les meilleurs performances.

7.3 Dataset

Afin d'être correctement exploités, les jeux de données doivent être enregistrés en tant que dataset dans le framework Microsoft Azure Machine Learning. Nous utilisons alors un stockage de type Blob, adapté à notre type de données non structurée.



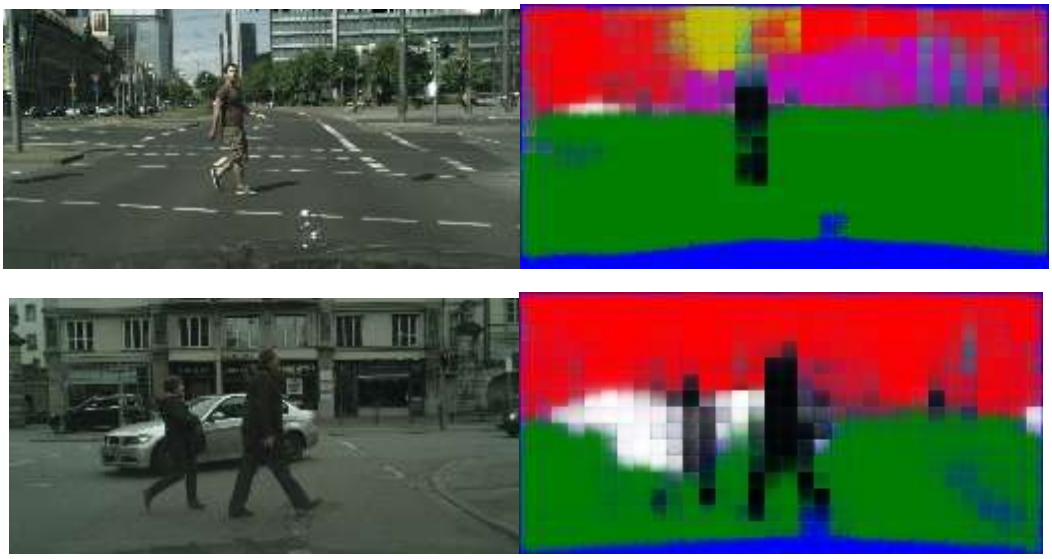
Enregistrement des datasets dans le framework Azure ML

7.4 Ressource de calcul

Une ressource de type GPU – Standard NC12 disposant de 112Gb de RAM a été utilisée pour l'entraînement final du modèle, sur le dataset entier. Grâce à cette puissance de calcul, le modèle final a été entraîné en 4h.

8. Résultats finaux

Après entraînement, le modèle présente un coefficient IoU de 0,6912, dépassant les performances des précédents modèles. Sur un échantillon d'image d'entrée, le modèle présente une capacité intéressante à identifier correctement la classe 'human' qui est la classe la plus critique du dataset pour des raisons évidentes de sécurité.



Exemple de masques prédits par le modèle identifiant la classe 'human'

9. Déploiement sur plateforme Azure

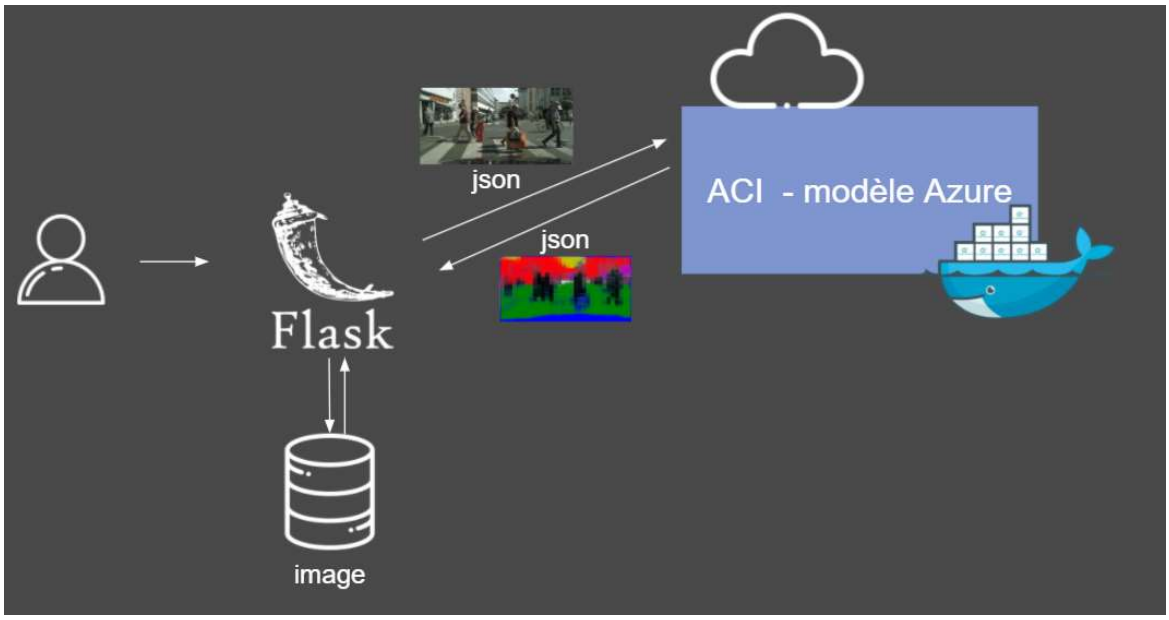
9.1 Azure ACI

Afin de pouvoir consommer le modèle et de servir le moteur d'inférence, nous avons déployé le modèle en utilisant une ACI Azure. L'ACI est un service managé qui permet de déployer directement un container sur le cloud sans utilisation de machine virtuelle. L'ACI se comporte alors comme une API à laquelle nous faisons appel en utilisant son URI.

9.2 Flask Front End

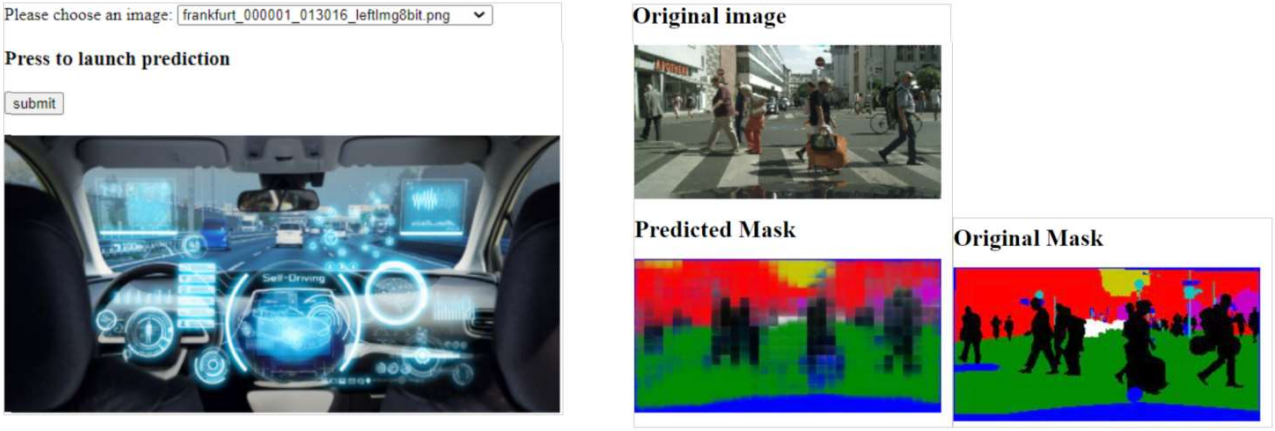
Pour fournir une interface utilisateur à titre d'illustration et de commodité d'utilisation, une application web élémentaire à été développée sur le framework Flask à été. Cette application se charge de la sérialisation/déserialisation des données JSON d'entrée et de sortie de l'ACI, mais également de l'affichage des résultats.

9.3 Architecture



Architecture du déploiement proposée

9.4 Interface Graphique



Interface graphique permettant l'appelle de l'ACI et l'affichage des résultats

10. Piste d'amélioration

Ce prototype à été réalisé et mis en œuvre dans les contraintes de temps et de ressources définies. Cependant, pour aller plus loin, nous suggérons les pistes suivantes (liste non-exhaustive) :

10.1 Prise en compte des déséquilibres inter classe

La métrique utilisée prend en compte le déséquilibre intra classe, cependant les classes ne sont pas toutes également représentées. Par conséquent nous suggérons d'implémenter une métrique /

fonction perte qui prenne en compte ce déséquilibre, en pondérant chaque classe par un coefficient. Par exemple, weighted IoU ou Focal Loss Entropy pourraient être implémentées. Ceci permettrait de contraindre le modèle à apprendre sur les classes sous représentées.

10.2 Travail en haute résolution

Bien que le jeu de données contiennent des images en haute résolution, nous avons choisi de travailler avec des images de 128x256 pixel afin de réduire les temps de calculs. Refaire le travail en haute résolution permettrait au modèle de pouvoir segmenter avec plus de détails mais avec un coût à l'entraînement et à l'inférence. Ce dimensionnement est étroitement lié au condition de déploiement final.

10.3 Différentes architectures

Nous n'avons utilisé que deux architectures alors que d'autres peuvent être implémentées comme Segnet,Resnet, PSPnEt et bien d'autres. Chaque architecture présente ses points forts et points faibles et sera plus ou moins bien adapté à notre problème.

10.4 Optimisation des hyperparamètres

Lors des phases préliminaires d'entraînement, une phase succincte d'optimisation d' hyperparamètres à été effectuée, nous suggérons fortement d'investiguer plus en avant cette optimisation, en terme de learning rate, type d'optimizer mais également de la fonction de perte en prenant en compte les différentes métriques.

10.5 Problématique temps réel

La finalité du modèle est d'être utilisé en temps réel. Dans le cahier des charges comme dans le prototype qui en découle, cette contrainte n'a pas été prise en compte, tant au niveau des performances de modèle en inférence qu'en terme architectural de la solution. Pour la suite du projet, il nous semble critique d'intégrer ces paramètres