Note technique

Romain Le Goff

Participez à la conception d'une voiture autonome

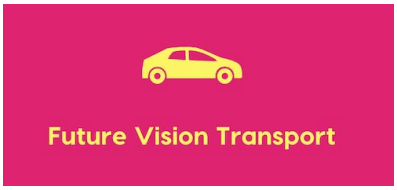


Table des matières

I. Introduction

Future Vision Transport est une entreprise qui conçoit des systèmes embarqués de vision par ordinateur pour les véhicules autonomes.

Je suis l'un des ingénieurs IA au sein de l'équipe R&D de cette entreprise. L'équipe est composée d'ingénieurs aux profils variés. Chacun des membres de l'équipe est spécialisé sur une des parties du système embarqué de vision par ordinateur.

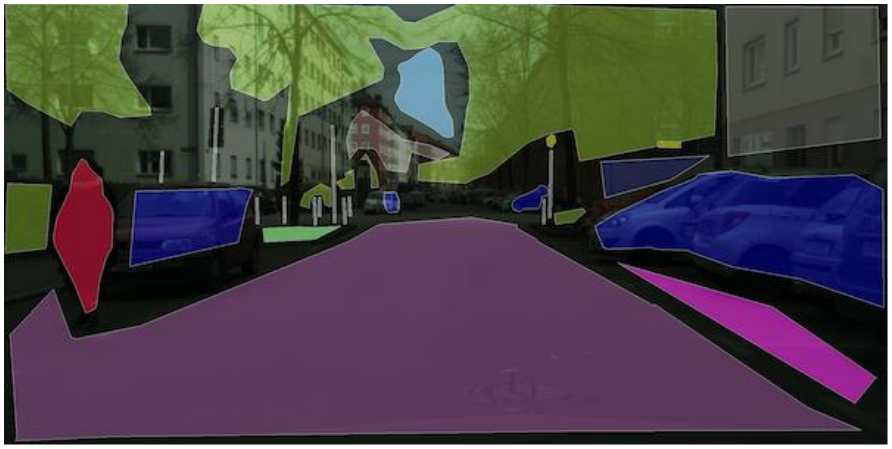
Voici les différentes parties du système :

- Acquisition des images en temps réel
- Traitement des images
- **Segmentation des images**
- Système de décision

Je travaille ici sur la partie « Segmentation des images », mon rôle est de concevoir un premier **modèle de segmentation d'images** qui devra s'intégrer facilement dans la chaîne complète du système embarqué.

Mon plan d'action est le suivant :

- **Entraîner un modèle de segmentation** des images sur **8 catégories principales** à partir du jeu de données de « Cityscapes Dataset »
- Concevoir une **API de prédiction**, et une **application web Flask** de présentation des résultats, qui seront déployés sur le Cloud.



II. Jeu de données

A. Analyse

Le jeu de données provient du site « Cityscapes Dataset ». Il est composé de 2 dossiers :

- **leftImg8bit** : contient les images RGB d'origine (données d'entrée)
- **gtFine** : contient les masques de la segmentation d'images (données de sortie)

Pour les données d'entrée, il s'agit d'images de dimension 2048 x 1024 pixels, sur 3 canaux (RGB)

Pour les données de sortie, il y a plusieurs fichiers. On ne va ici s'intéresser qu'aux fichiers « _labelIds.png », qui correspondent aux masques des images RGB, encodés avec les IDs des différents labels de segmentation (32 catégories).

Les fichiers « _color.png » représentent le masque avec un code couleur RGB pour chaque label :

Exemple :

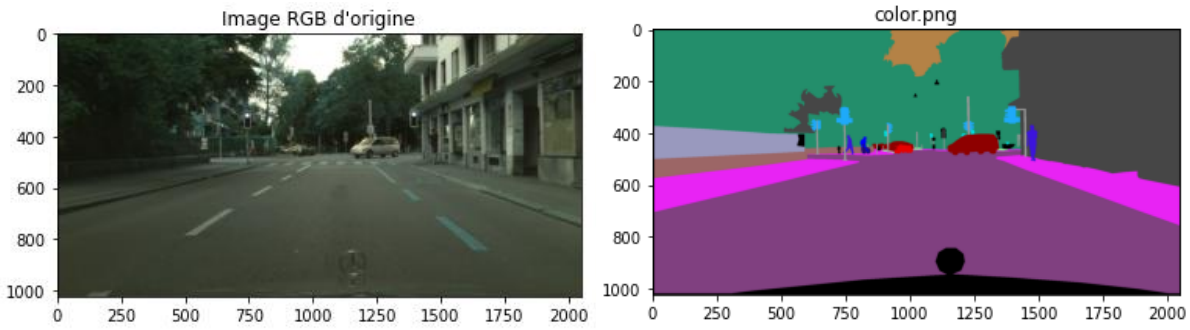


Figure 1 : Image RGB d'origine et son masque

Le jeu de données est également séparé en 3 parties, un jeu de test, un jeu d'entraînement et un jeu de validation.

- **Train** : 2975 images
- **Test** : 1525 images
- **Val** : 500 images

B. Catégorisation

Mon objectif est d'entraîner un modèle sur **8 catégories**. En récupérant la documentation du jeu de données, j'ai pu déterminer à quelle catégorie principale correspond chaque ID des 32 sous catégories :

- **Vide** : [0, 1, 2, 3, 4, 5, 6]
- **Route** : [7, 8, 9, 10]
- **Construction** : [11, 12, 13, 14, 15, 16]
- **Objet** : [17, 18, 19, 20]
- **Nature** : [21, 22]
- **Ciel** : [23]
- **Humain** : [24, 25]
- **Véhicule** : [26, 27, 28, 29, 30, 31, 32, 33, -1]

Par mapping via ce tableau, je peux en déduire chacun des masques d'origine sur 8 catégories.

Exemple :

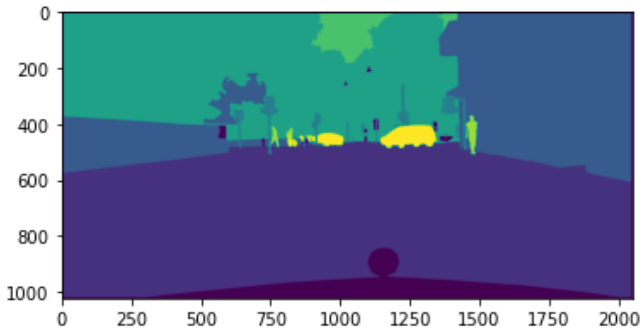


Figure 2 : Masque d'origine sur 8 catégories

III. Etat de l'art

Je vais ici présenter les modèles qui seront utilisés et entraînés, ils sont tous basés sur **U-NET** qui est un réseau de neurones à convolution.

A. U-NET et variantes

1. U-NET

U-NET est un modèle de réseau de neurones dédié aux taches de Vision par Ordinateur (Computer Vision) et plus particulièrement aux problèmes de Segmentation Sémantique.

L'architecture de U-NET est composée de deux « chemins ». Le premier est le chemin de contraction, aussi appelé **encodeur**. Il est utilisé pour capturer le contexte d'une image.

Il s'agit en fait d'un **assemblage de couches de convolution** et de couches de « max pooling » permettant de créer une carte de caractéristiques d'une image et de réduire sa taille pour diminuer le nombre de paramètres du réseau.

Le second chemin est celui de l'expansion symétrique, aussi appelé **décodeur**. Il permet la localisation précise grâce à la convolution transposée et permet également de retrouver la taille initiale de l'image.

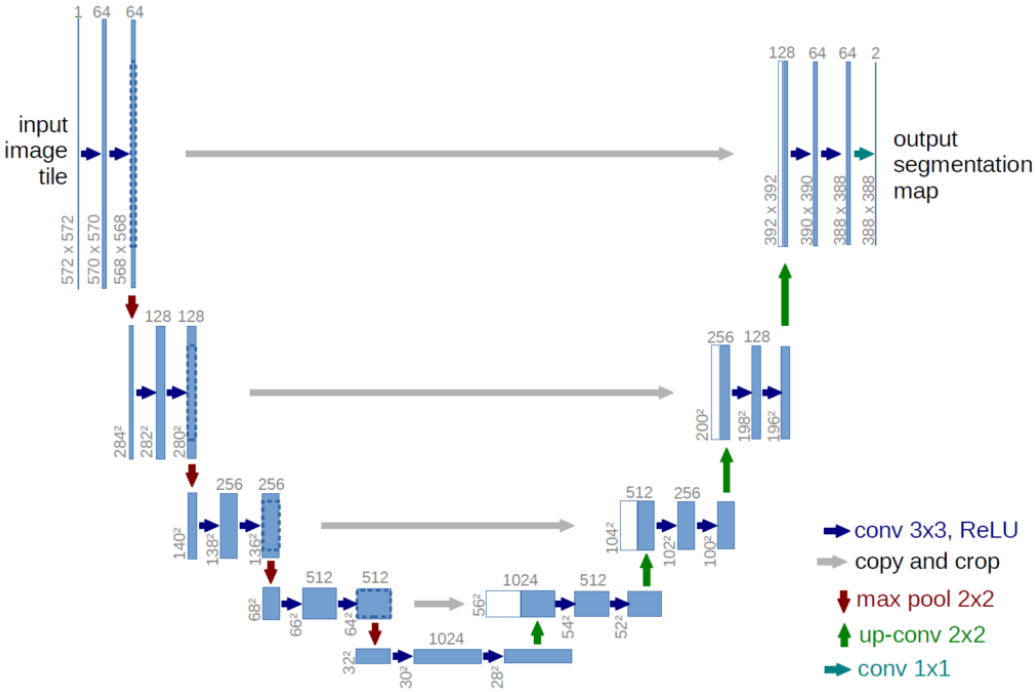


Figure 3 : Architecture U-NET

2. Variantes

J'utiliserais également des variantes de U-NET, on vient remplacer la partie encodeur par un autre type de réseau de neurones.

- **Restnet-50** : c'est un réseau de neurones à convolution avec 50 couches de profondeur.

- **VGG16-UNET** : pour la partie encodeur on utilisera ici VGG16 qui est un autre type de réseau de neurones à convolution.

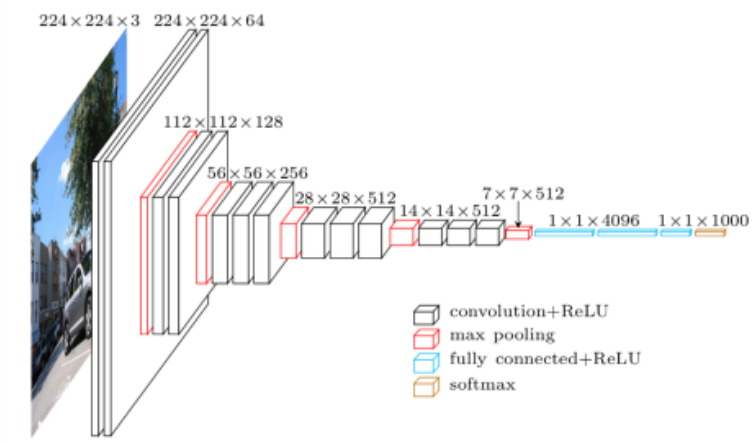


Figure 4 : Architecture VGG16

IV. Stratégie d'apprentissage

A. Métriques

Pour la métrique j'ai utilisé l'indice de Jaccard ou **Mean IoU (Moyenne Intersection over Union),** qui répond bien à une problématique de segmentation multi-classes.

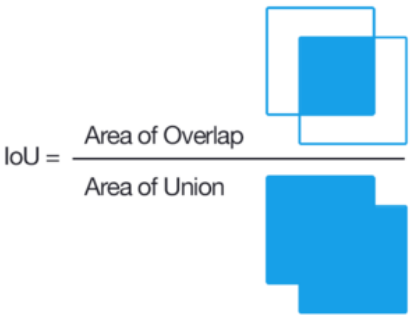


Figure 5 : calcul IoU

IoU est une métrique permettant d'obtenir un score basé sur la bonne prédiction d'une classe.

Pour une segmentation multi-class, Mean IoU est calculé en prenant l'IoU de chaque classe et en faisant la moyenne.

Pour chaque modèle, les **temps d'entrainement et de prédiction** seront également enregistrés.

Les images sont également transformées avant d'être utilisées par les modèles, la taille de chaque **image est réduite à 256 x 256 pixels.**

B. Baseline

Chaque modèle sera comparé à un modèle simple qui servira de référence, afin de les comparer et observer une éventuelle amélioration.

Le **modèle simple est U-Net**, sans augmentation de donnés, avec la fonction loss "categorical_crossentropy" (voir plus loin).

C. Fonctions Loss

Je vais **optimiser la fonction loss** qui est un hyperparamètre en entraînant plusieurs fois le modèle simple U-Net, afin de déterminer la meilleure fonction loss et l'utiliser sur les modèles suivants.

Les différentes fonctions loss utilisées :

- **Categorical_crossentropy** : Quantifie la différence entre 2 distributions de probabilité.
- **Dice_loss** : 1 – dice_coeff (qui correspond à 2 * intersection / union)
- **Combine_loss** : somme de 2 fonctions loss : categorical_crossentropy + 3*dice_loss

D. Data augmentation

Afin d'améliorer la modélisation, je vais **augmenter les données**, c'est à dire compléter le jeu de données **en transformant les images**, soit géométriquement, soit en rajoutant du bruitage, ou les deux.

Lorsque la data augmentation est sélectionnée lors de l'entraînement d'un modèle, le nombre d'images et doublé, chacune étant augmentée. Voici les différentes transformations appliquées aléatoirement :

- Symétrie verticale
- Flou gaussien
- Modification du contraste
- Ajout d'un bruit gaussien
- Modification de la luminosité
- Zoom, translation, rotation

Exemples :

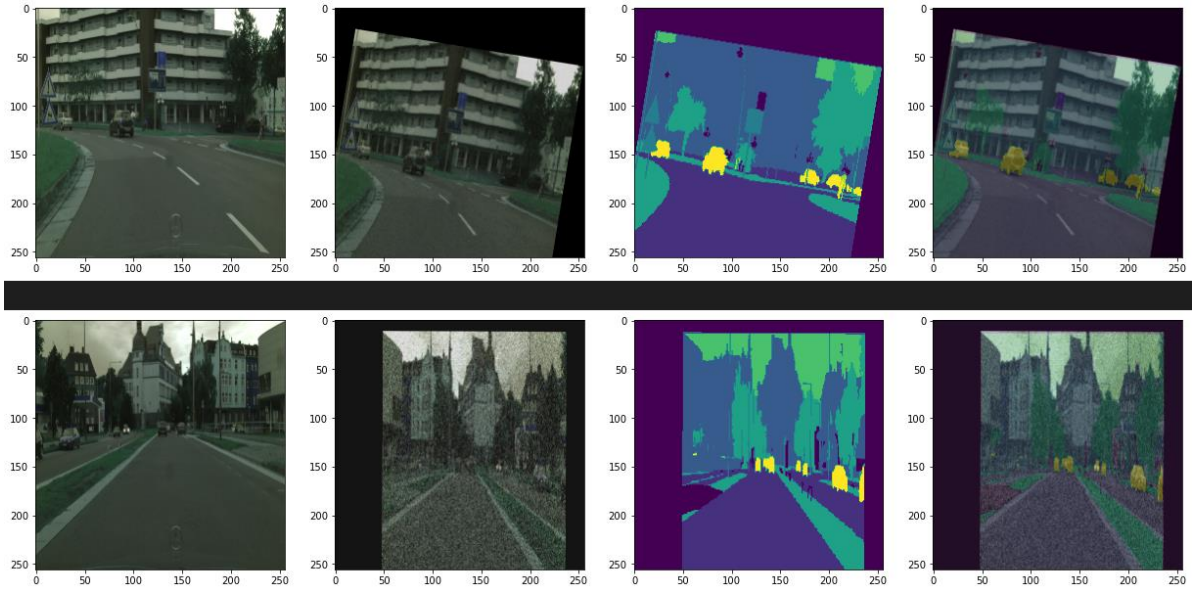


Figure 6 : Exemples de data augmentation (images et masques)

E. Générateurs de données

Le jeu de données étant volumineux, je ne peux pas l'utiliser dans son intégralité d'un seul coup, je serais confronté à un problème de mémoire.

La solution est d'utiliser un **Générateur de données**, qui permet de déclarer une fonction qui agît comme un **itérateur dans une boucle**.

Il vient charger les fichiers par paquet.

J'ai implémenté 2 générateurs :

- Un premier **classique** qui vient récupérer à la volée les images et masques **par paquet de 20**
- Un second qui fonctionne de la même manière mais qui en plus vient appliquer la **data augmentation** sur chaque image.

V. Résultats

Pour chaque modèle entraîné, j'affiche un graphe présentant l'évolution de la fonction loss utilisée et de la métrique, pour l'échantillon d'entraînement et celui de validation. (Voir exemple plus loin)

Liste des modèles entraînés :

- **U-NET**, *loss* : category_crossentropy, *data augmentation* : False
- **U-NET**, *loss* : dice_loss, *data augmentation* : False
- **U-NET**, *loss* : combine_loss, *data augmentation* : False
- **U-NET**, *loss* : combine_loss, *data augmentation* : True
- **Resnet-50**, *loss* : combine_loss, *data augmentation* : True
- **VGG16-U-NET**, *loss* : combine_loss, *data augmentation* : False
- **VGG16-U-NET**, *loss* : combine_loss, *data augmentation* : True
- **VGG16-U-NET**, *loss* : combine_loss, *freeze des couches*
-

L'optimisation de la fonction loss sur le modèle simple a permis de déterminer que la meilleure était la fonction « **combine_loss** », c'est pour cette raison qu'elle est utilisée par la suite pour chacun des modèles.

Chacun des modèles a été entraîné sur **50 epochs**, avec une stratégie **d'Early stopping** si la métrique ne s'améliore pas après un certains nombres d'epochs.

Voici un exemple d'évolution de la métrique et de la fonction loss, sur le modèle **VGG16-U-NET**, *loss* : combine_loss, *data augmentation* : True, qui s'avère être le meilleur modèle (voir tableau de synthèse).

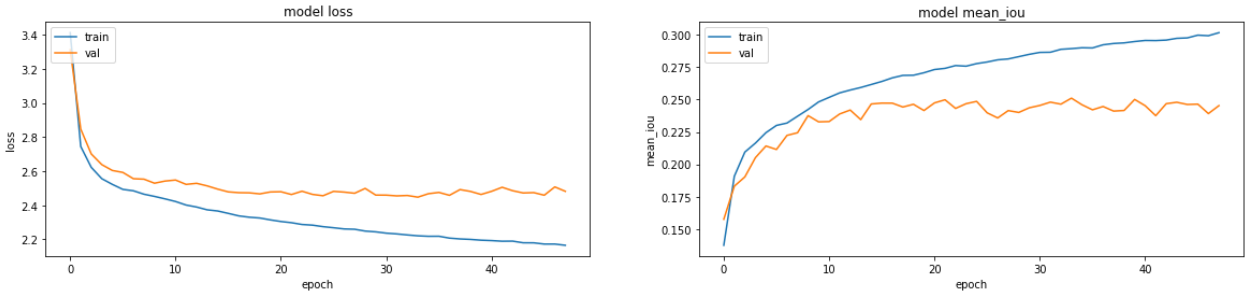


Figure 7 : courbes d'apprentissage

Sur les données de test, on obtient avec ce modèle :

- **Mean_IoU** : 0,3167
- **Loss** : 2.281

J'affiche également pour chacun des modèles quelques exemples de résultats de segmentation, afin de visualiser les différences.

Ce modèle est celui qui segmente le mieux les catégories les plus visibles comme le ciel, la nature, les véhicules et la route.

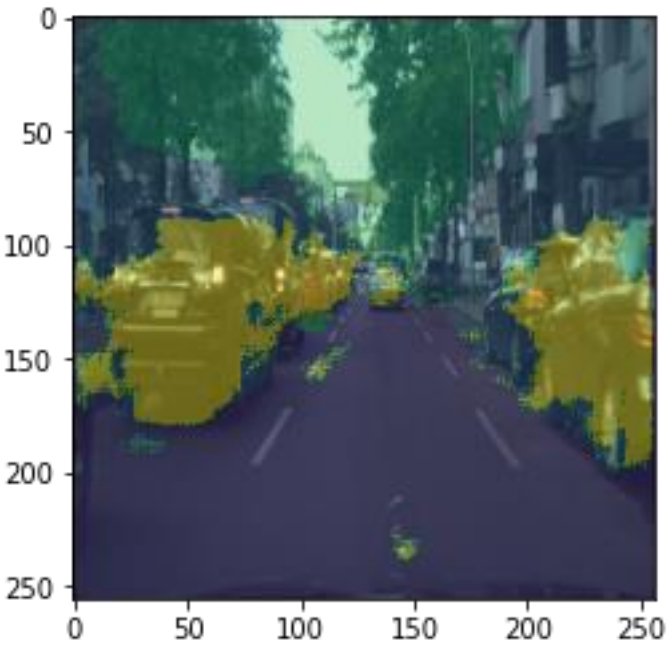


Figure 8 : Exemple de segmentation avec ce modèle

Synthèse comparative :

	loss type	loss score	mean_iou	Training time	Predict time
U-Net_base	categorical_crossentropy	1.045049	0.302060	6005.426807	1.509681
U-Net_dice_loss	dice_loss	0.376354	0.293561	11075.384537	0.595256
U-Net_combine_loss	combine_loss	2.482293	0.302627	8733.732015	0.608075
U-Net_augmented	combine_loss	2.330191	0.303681	9423.844568	0.646507
ResNet50_U-Net_augmented	combine_loss	2.311371	0.313203	12474.215919	1.753642
VGG16_U-Net_base	combine_loss	2.267051	0.325019	8658.989292	2.976111
VGG16_U-Net_augmented	combine_loss	2.298635	0.319571	10783.304042	0.527344
VGG16_U-Net_freeze	combine_loss	2.260754	0.319250	9002.626294	0.582971

Figure 9 : Synthèse comparative

Le modèle le plus performant est donc le **VGG16-U-NET**, avec augmentation de données.

On observe visuellement que l'augmentation de données améliore la précision de segmentation sur les catégories les plus visibles même si le Mean-IoU est presque identique. L'erreur est simplement plus présente sur les catégories moins visibles ce qui équilibre le score entre tous les modèles.

VI. Déploiement du modèle

Une fois le modèle le plus performant sélectionné et sauvegardé, je vais le charger dans une application Flask que je vais déployer sur le cloud, en utilisant Docker et Azure.

A. Application Flask

Flask est un **framework de développement web** en Python.

Je vais pouvoir créer une application web permettant de présenter les résultats en choisissant une image parmi une liste, qui sera ensuite segmentée en utilisant le modèle choisi.

Je vais également **exposer une API** qui peut être consommée par un autre service.

B. Docker

Docker est une technologie de conteneurisation qui facilite la gestion de dépendance au sein d'un projet.

Il existe trois concepts clés dans Docker : **les conteneurs, les images et les fichiers Docker (Dockerfile).**

- Un **conteneur** est un espace dans lequel une application tourne avec son propre environnement. Chaque conteneur est une instance d'une **image**.
- Les **images** représentent le contexte que plusieurs conteneurs peuvent exécuter.
- Un **Dockerfile** est un fichier qui liste les instructions à exécuter pour build une image. Il est lu de haut en bas au cours du processus de build.
-

Figure 10 : Processus Docker

Le **Dockerfile** permet de créer une image. Cette **image** contient la liste des instructions qu'un **conteneur** devra exécuter lorsqu'il sera créé à partir de cette même image.

C. Azure

Une fois le conteneur crée, je vais le déployer sur **Azure** dans la partie **App Service**, en tant qu'application Web, afin de la rendre accessible à n'importe qui.

Le déploiement peut se faire avec les extensions Docker et Azure directement depuis Visual Studio Code, après avoir créé et configuré un compte sur les deux plateformes.

D. Tests

Une fois le déploiement effectué, l'application web est accessible à l'adresse suivante : https://ocp8-segmentation.azurewebsites.net/

Pour tester la segmentation, il y a au choix 20 images avec le masque d'origine associé. Le résultat s'affiche après avoir cliqué sur le bouton soumettre.

Exemple :

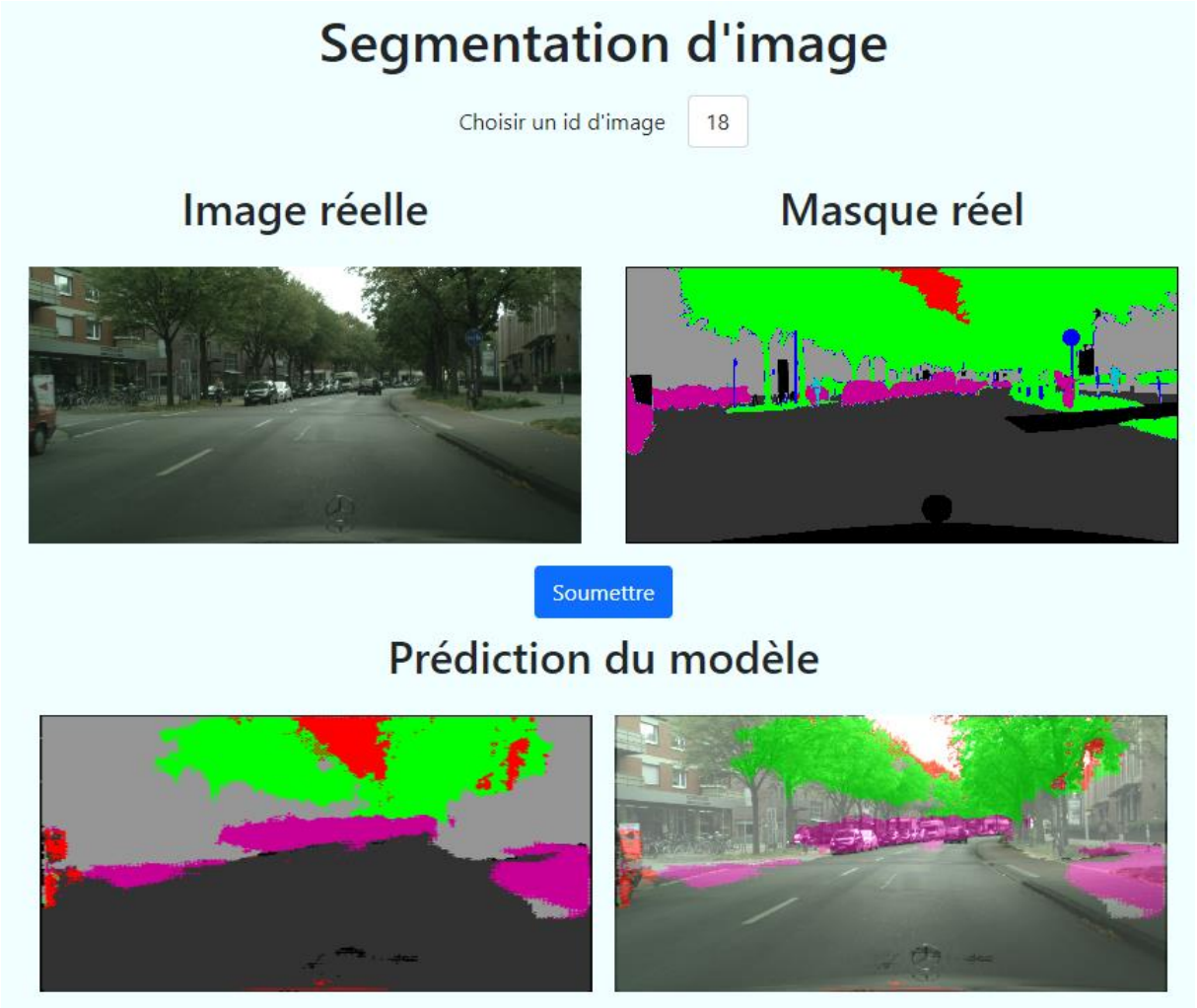


Figure 11 : Utilisation de l'application flask déployée sur le cloud

Une API est également exposée, permettant d'être consommée par n'importe quel service, cette fois ci n'importe quelle image peut être segmentée.

L'adresse du **endpoint** est la suivante :

https://ocp8-segmentation.azurewebsites.net/segment/

L'API prend en entrée une image (à placer dans le body de la requête), et retourne l'image segmentée. On va utiliser la même image sélectionnée précédemment pour montrer que la segmentation est identique.

Exemple :

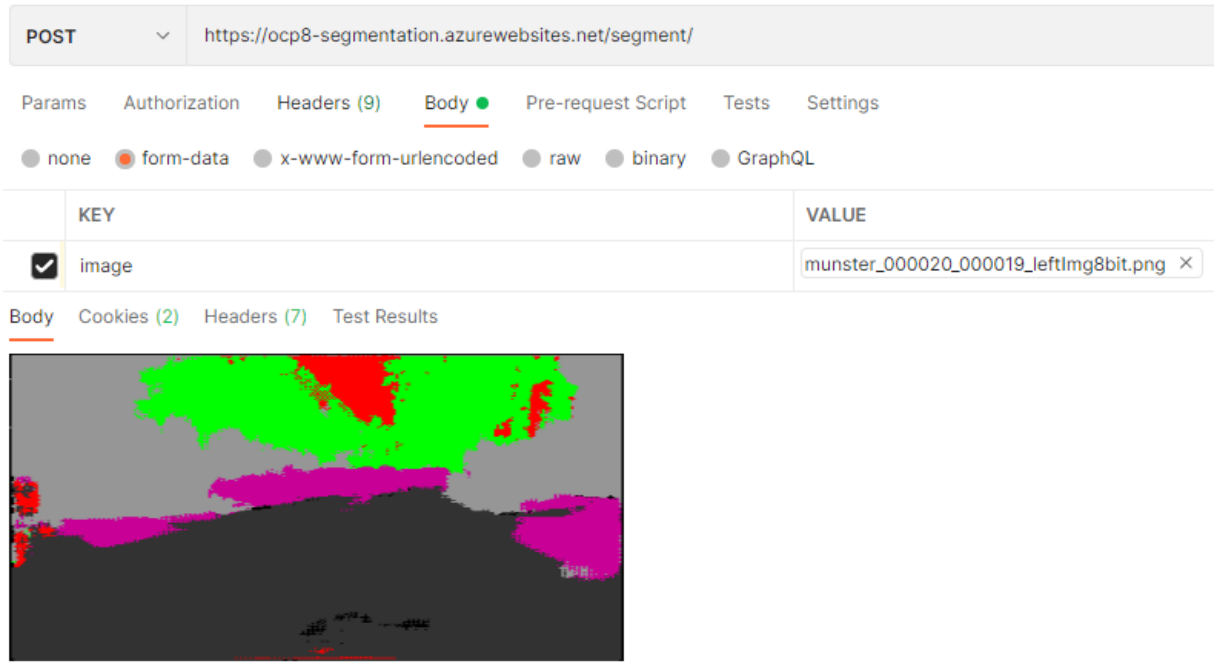


Figure 12 : Test du endpoint via Postman

VII. Conclusion

Le meilleur modèle que j'ai sélectionné dans le cadre de ce projet est :

VGG16-U-NET avec **augmentation de données**, et fonction loss **Combine_loss**

Même si l'on se rapproche de la segmentation initiale, celle-ci est loin d'être optimale. Et en fonction des images, il peut y avoir beaucoup d'erreurs.

(Le score Mean IoU est de seulement 0.3167)

Amélioration du modèle :

Afin d'augmenter ce score, plusieurs options sont envisageables :

- Avoir plus d'images dans le jeu de données
- Moins réduire les images en entrée du modèle (actuellement 256 x 256)
- Trouver des modèles plus performants
- Optimiser d'autres hyperparamètres comme : le learning rate, l'optimizer, le nombre d'epochs etc ...
- Améliorer l'augmentation de données
- Augmenter le nombre de batchs dans l'utilisation du Générateur

Tout ceci rallonge la durée d'entraînement du modèle, et demande une machine plus puissante.

Pour une voiture autonome ce modèle n'est en l'état pas suffisant, par son manque de précision, et par son temps de prédiction qui est trop long. La voiture a besoin d'un temps de prédiction très court pour dépasser le réflexe humain et avoir un intérêt. Il faudrait donc changer de modèle et augmenter la puissance de calcul.

Amélioration de l'API et de l'application Web :

L'interface pourrait être amélioré et proposer d'autres options, comme par exemple la sélection du modèle.

Le déploiement pourrait être automatisé avec une approche CI/CD. Lors d'un changement de version de l'application, celui-ci serait détecté (via un dépôt Github par exemple), et la conteneurisation et déploiement sur Azure seraient relancés automatiquement.