

Segmentation d'images

Contexte et objectifs

Future Vision Transport est une entreprise qui conçoit des systèmes embarqués de vision par ordinateur pour les véhicules autonomes. Le système embarqué créé par l'équipe de R&D est composé de quatre parties :

1. Acquisition des images en temps réel
2. Traitement des images
3. Segmentation des images
4. Système de décision

Cette note technique décrit le processus de segmentation des images issue du traitement précédent et du transfert vers le système de décision.

Table des matières

Contexte et objectifs.....	1
État de l'art.....	3
Modèle de segmentation sémantique.....	4
Fully Convolutional Networks (FCN)	4
U-Net	5
PSPNet	6
Exploration des modèles.....	7
Calcul de la perte	8
Entropie croisée catégorielle (Categorical Cross Entropy)	8
Métriques	8
Précision	8
Coefficient IoU	8
Coefficient de Dice	8
Modèles	9
FCN-8.....	9
U-Net.....	10
PSPNet	11
Synthèse des différents modèles.....	12
Modèle U-Net.....	13
Dice loss	13
Prédiction	13
Data augmentation	13
Prédiction	14
Synthèse de l'utilisation du modèle U-Net	14
Conclusions.....	15
Annexes.....	16
Déploiement Azure	16
Déploiement du modèle choisi.....	16
Création du jeu de données	16
Création de l'API.....	17
Développement de l'API.....	18
Rendu de l'API	19
Références.....	20

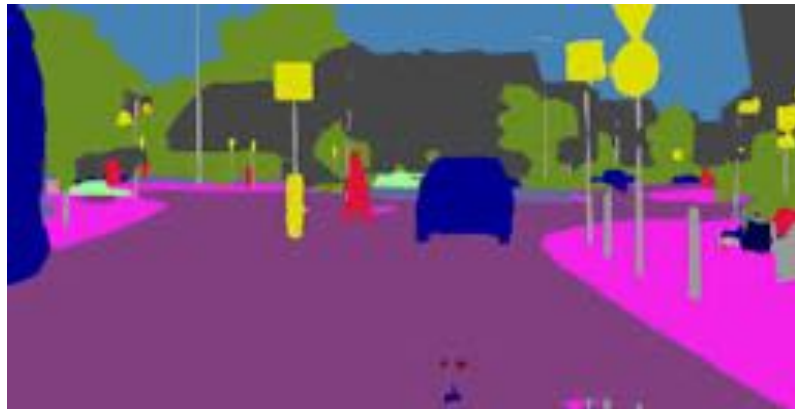
État de l'art

La segmentation d'images est l'étape préliminaire à la reconnaissance d'objets. Cela va consister à extraire les éléments constituant une image et ainsi partitionner l'image en un certain nombre de régions. Les régions sont déterminées par des critères d'homogénéité, tel que le niveau de gris ou la couleur.

Il existe deux types de segmentation d'images (sémantique et d'instance), chacun ayant des objectifs différents.

L'objectif de la **segmentation sémantique** d'images est d'étiqueter chaque pixel d'une image avec une classe correspondante de ce qui est représenté. Il est important

de noter que cette segmentation ne s'intéresse qu'à la catégorie du pixel, c'est à dire les objets d'une même classe ne seront pas séparés. La segmentation sémantique est très utilisée pour les voitures autonomes et dans l'imagerie médicale



Dans l'exemple ci-dessous les humains sont en rouge, les véhicules en bleu et les panneaux de signalisation en jaune.

Les modèles de **segmentation d'instance** vont distinguer les objets distincts au sein d'une même classe. Dans cet exemple chaque véhicule est identifié avec une couleur différente.

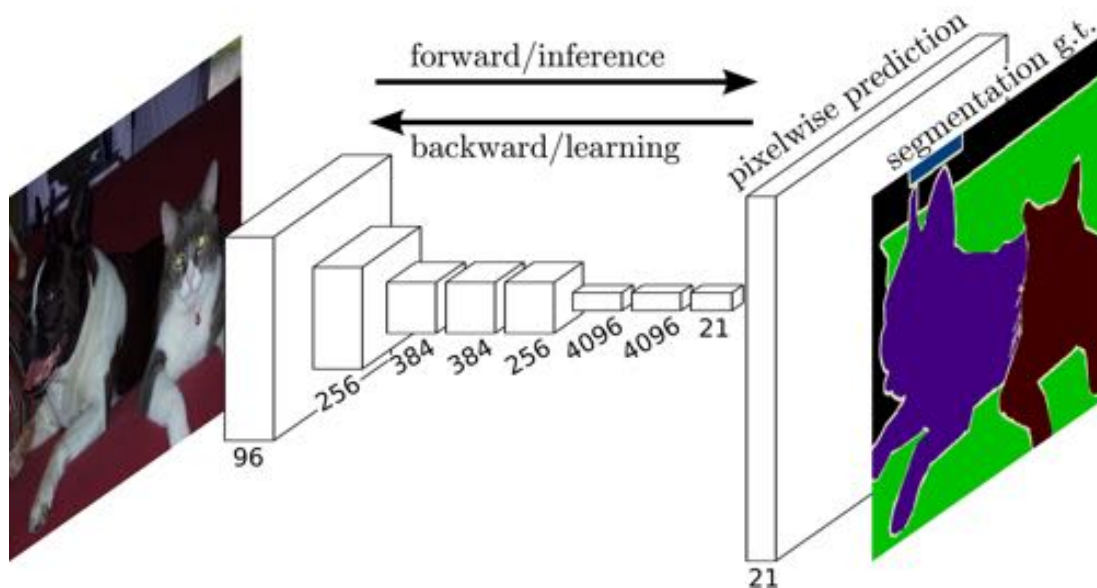


Modèle de segmentation sémantique

Il existe actuellement plus de 20 modèles de segmentation sémantique. Parmi les architectures les plus populaires il y a les modèles Fully Convolutional Networks, U-Net et PSPNet

Fully Convolutional Networks (FCN)

Les réseaux neuronaux entièrement convolutifs ont été proposés pour la première fois par Long, Shelhamer et Darrell dans leur article intitulé "[Fully Convolutional Neural Networks for Semantic Segmentation](#)". Leur objectif était de remplacer les couches entièrement connectées et les CNNs typiques par des couches convolutives qui agissent comme un décodeur.

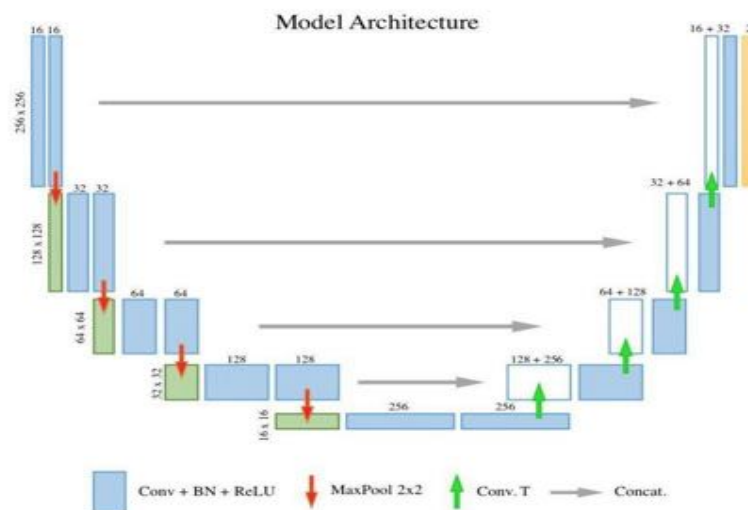


Ce modèle est divisé en deux parties, la première l'encodeur (qui souvent utilise un modèle pré-entraîné), va détecter les caractéristiques et réduire l'échelle de l'image, tandis que la deuxième partie va augmenter l'échelle de l'image et créer le masque-

Ce modèle est très efficace pour la segmentation d'images mais un inconvénient de ce modèle est l'utilisation des données pré-entraînées qui force la taille des images à un rapport d'aspect de 1 (carré), alors que les images des voitures autonomes sont plutôt avec un rapport de 1/2.

U-Net

Le modèle U-net a été proposé pour la première fois par : Olaf Ronneberger, Philipp Fischer et Thomas Brox dans leur article intitulé "[U-Net: Convolutional Networks for Biomedical Image Segmentation](#)". Le modèle U-net est construit sur le modèle FCN-8 pour s'adapter à l'imagerie médicale.

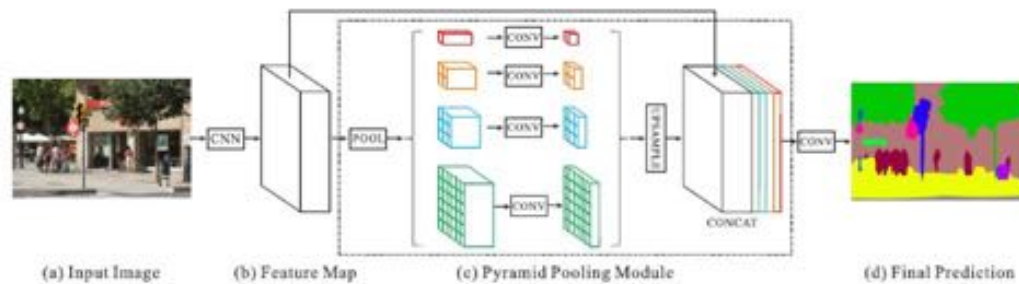


C'est un modèle symétrique en forme de U séparé par un bottleneck. La première partie appelée contraction ou downsampling effectue l'extraction des caractéristiques contextuelles, la deuxième partie est le bottleneck qui force le modèle à apprendre d'une compression des données d'entrée et la dernière partie (qui est l'inverse exact de la première partie) appelée expansion ou upsampling effectue la localisation des caractéristiques.

Un des avantages de ce modèle c'est qu'il est plus rapide qu'un modèle FCN.

PSPNet

Le modèle U-net a été proposé pour la première fois par : Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang et Jiaya Jia dans leur article intitulé "[Pyramid Scene Parsing Network](#)"



H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia. Pyramid Scene Parsing Network. In CVPR, 2017.

Ce modèle est divisé en trois parties ; la première partie effectue l'extraction des caractéristiques à partir d'un CNN pré-entraîné, puis la deuxième partie (Pyramid Pooling Module) est constituée de 2 pyramides de 4 niveaux qui vont effectuer des convolutions, enfin la dernière partie est une couche de convolution qui va permettre de faire les prédictions.

Exploration des modèles

Pour l'étude de la segmentation des images nous avons utilisé les données du site [CityScapes](#) qui propose des scènes de rue urbaines issues de caméra embarquée. Ce jeu de données est diversifié car il inclut : les saisons, les villes, l'heure, les conditions météo, etc.

Pour l'entraînement des modèles nous avons sélectionné un jeu de données de 2750 images des villes de : Aachen, Bochum, Bremen, Cologne, Darmstadt, Dusseldorf, Erfurt, Hamburg, Hanover, Jena, Krefeld, Mönchengladbach, Strasbourg, Stuttgart, Tübingen, Ulm, Weimar et Zurich.

Les masques de **cityscapes** sont annotés avec 30 classes, nous avons réduit ces classes à huit en les regroupant comme suit :

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

Avec la palette de couleur :



Les images d'origine sont au format 1024x2048 pixels et sont réduites en 128x256 pixels.

Calcul de la perte

Entropie croisée catégorielle (Categorical Cross Entropy)

La perte d'entropie croisée catégorielle (ou log-vraisemblance négative), mesure la similarité entre les données prédites et les données vraies.

Métriques

Pour chaque modèle nous avons utilisé comme métrique la précision, le coefficient IoU et le coefficient Dice

Précision

La précision est la proportion des prédictions pertinentes parmi l'ensemble des données proposées

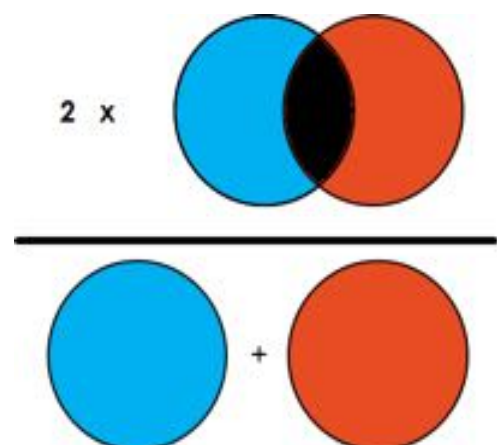


Coefficient IoU

Le coefficient IoU (Intersection-Over-Union), également connu sous le nom d'indice de Jaquard est une métrique couramment utilisée pour la segmentation sémantique. L'IoU est la zone de chevauchement entre la segmentation prédite et la vérité terrain divisée par la zone d'union entre la segmentation prédite et la vérité terrain

Coefficient de Dice

Le coefficient de Dice (également connu sous le nom de coefficient de Sørensen-Dice et de score F1) est défini comme le double de la surface de l'intersection de A et B, divisé par la somme des surfaces de A et B :



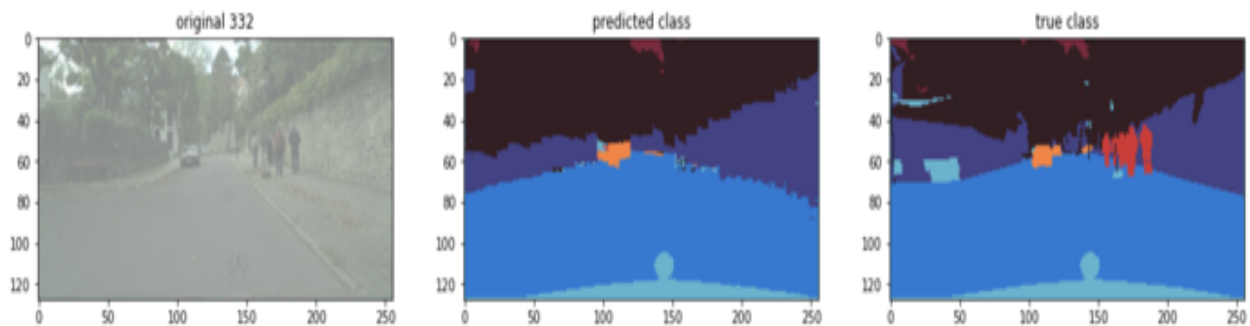
Modèles

FCN-8

Pour le modèle FCN-8, nous avons utilisé un modèle pré-entraîné VGG16. Les sources sont disponibles dans le fichier P8_01_Model_VGG16FCN8.ipynb

Temps d'entrainement	Perte	Précision	IoU	Dice
00:31:04	0.584	0.849	0.680	0.805

Prédiction



IoU

Catégorie	IoU
void	0.444
flat	0.491
construction	0.389
object	0.000
nature	0.460
sky	0.347
human	0.000
vehicle	0.325

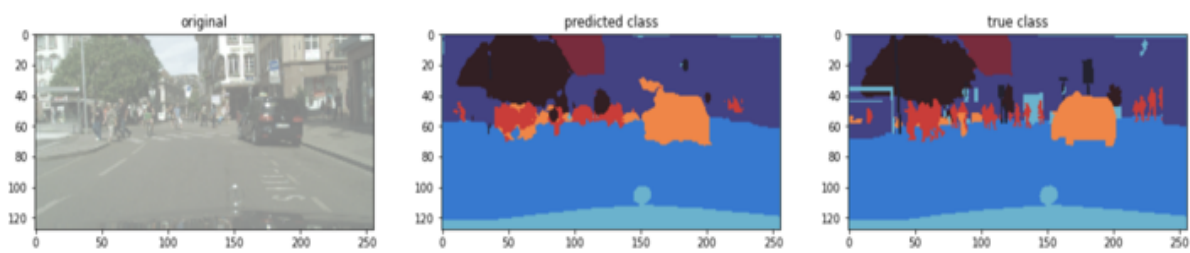
Ce modèle montre de bonnes performances en ce qui concerne la catégorie "Flat" (on la retrouve sur toutes les images), mais présente des lacunes pour la catégorie "Human".

U-Net

Les sources sont disponibles dans le fichier P8_02_Model_UNet.ipynb

Temps d'entrainement	Perte	Précision	IoU	Dice
00:11:27	0.452	0.885	0.297	0.457

Prédiction



IoU

Catégorie	IoU
void	0.458
flat	0.487
construction	0.433
object	0.092
nature	0.447
sky	0.492
human	0.301
vehicle	0.398

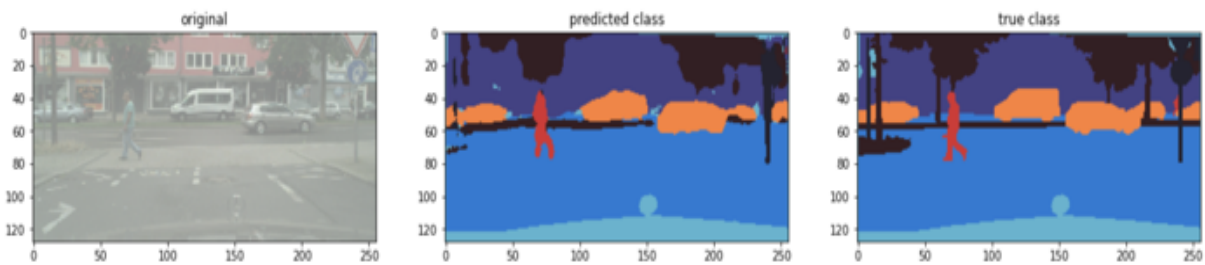
Ce modèle montre de très bonnes performances en temps de traitement. Au contraire du modèle FCN la catégorie "Human" est mieux détectée.

PSPNet

Les sources sont disponibles dans le fichier P8_03_Model_PSPNet.ipynb

Temps d'entrainement	Perte	Précision	IoU	Dice
01:09:07	0.610	0.904	0.817	0.896

Prédiction



IoU

Catégorie	IoU
void	0.483
flat	0.483
construction	0.447
object	0.331
nature	0.413
sky	0.000
human	0.381
vehicle	0.436

Le modèle PSPNet donne de très bons résultats, mais le temps d'entrainement est très long.

Synthèse des différents modèles

Modèle	Temps d'entrainement	Perte	Précision	IoU	Dice
FCN-8	00:31:04	0.584	0.849	0.680	0.805
U-Net	00:11:27	0.452	0.885	0.297	0.457
PSP-Net	01:09:07	0.610	0.904	0.817	0.896

D'après les différents modèles utilisés, le modèle U-net semble plus approprié pour la segmentation que nous essayons de réaliser. Il combine un temps de traitement court et une précision plus grande et une faible perte, cependant les scores IoU et Dice sont faibles ce qui pourrait faire tendre à un moins bon modèle. Nous utiliserons ce modèle pour un prochain entraînement avec un changement de calcul de perte et d'augmentation des données.

Modèle U-Net

Le modèle U-Net ayant été choisi, nous avons procédé à des ajustements afin d'augmenter ses performances. Dans un premier temps nous avons utilisé un Dice Loss, puis nous avons procédé à une augmentation des données.

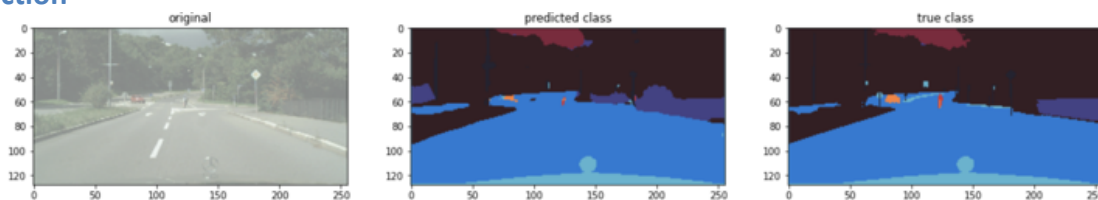
Dice loss

L'utilisation de l'entropie inter-catégorique comme calcul de perte ne donne pas de bon résultat car elle n'est pas assez précise au niveau de catégorisation des pixels. Le calcul de la perte avec le coefficient de Dice donne de meilleurs résultats avec des classes déséquilibrées.

Les sources sont disponibles dans le fichier P8_04_Model_UNet-Dice-loss.ipynb

Temps d'entrainement	Perte	Précision	IoU	Dice
00:11:27	0.452	0.885	0.297	0.457
00:11:44	0.067	0.887	0.770	0.876

Prédiction



PSPNet prediction

L'utilisation du calcul de la perte avec l'algorithme Dice permet d'augmenter la précision, ainsi que le score IoU et Dice.

Data augmentation

L'augmentation de données permet d'ajouter des images, relativement similaires aux images du jeu d'entrainement. En effet ses images seront modifiées en fonction de certains critères :

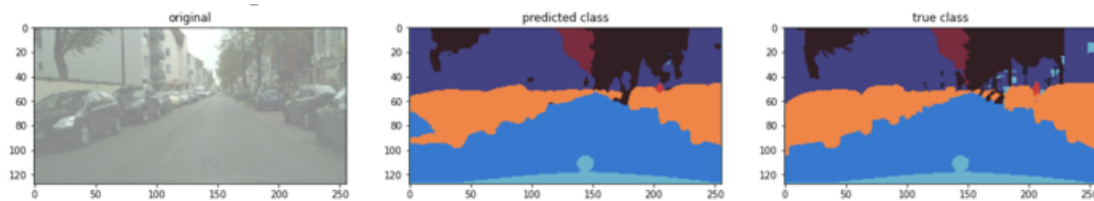
- Retournement horizontal
- Luminosité et contraste aléatoire
- Flou
- Rotation (max 45°)

Nous avons utilisé la bibliothèque [ImageDataAugmentor](#) qui permet de réaliser facilement des augmentations d'images. Cette bibliothèque supporte [ImageDataGenerator](#) de Keras et [albumentations](#).

Les sources sont disponibles dans le fichier P8_05_Model_UNet-Dice-loss-Augm.ipynb

Temps d'entrainement	Perte	Précision	IoU	Dice
00:11:27	0.452	0.885	0.297	0.457
00:10:06	0.063	0.889	0.735	0.844

Prédiction



PSPNet prediction

L'utilisation de données supplémentaires a permis d'augmenter les scores.

Synthèse de l'utilisation du modèle U-Net

Même si le modèle U-Net montrait de moindres performances par rapport au modèle PSPNet, l'utilisation d'un autre calcul de perte et l'ajout de données a permis d'obtenir un modèle plus performant.

Conclusions

Nous avons privilégié le modèle U-Net car celui-ci offre de bonnes performances et surtout une grande rapidité d'exécution, ce qui est primordial dans l'analyse des images d'une voiture autonome. Cependant nous avons vu que certaines catégories étaient difficiles à détecter, il nous semble que cela est dû à la finesse de l'objet et étant donné que nous réduisons l'image, il serait pertinent d'utiliser une résolution plus importante tout en gardant la performance de traitement.

De nouveaux modèles sont élaborés chaque année et il convient de suivre leurs évolutions. Comme le papier "Searching for MobileNetV3" de 2020 de Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam, qui proposent un nouveau décodeur de segmentation : Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP) et une augmentation de la vitesse d'exécution

Annexes

Déploiement Azure

Le modèle ayant été choisi et entraîné celui-ci est sauvegardé en local. Nous avons développé une API web afin de tester le modèle. Cette API prend en entrée l'identifiant d'une image et renvoie l'image elle-même, la segmentation prédite et la segmentation attendue.



Déploiement du modèle choisi

Le modèle est transféré dans Azure Machine Learning, afin qu'il puisse être utilisé par l'API. Pour cela nous avons utilisé la méthode « Register ».

```
model = Model.register(workspace = ws,
                        model_path= os.path.join(app_folder, 'save_model'),
                        model_name = 'future-vision-transport',
                        description = 'Semantic segmentation')
```

Les sources sont disponibles dans le fichier P8_06_Register-model.ipynb


Cette API étant exposée dans l'espace public nous avons ajouté une couche de sécurité en ajoutant un utilisateur spécifique pour l'accès au modèle et en utilisant la méthode « ServicePrincipalAuthentication » pour l'accès depuis l'API Flask.






<input type="checkbox"/>	Nom	Type	Rôle	Portée	Condition
Contributeur					
<input type="checkbox"/>	 futur-vehicule-transport	Application	Contributeur ⓘ	Abonnement (Hérité)	Aucun
<input type="checkbox"/>	 fvt	Application	Contributeur ⓘ	Cette ressource	Aucun


Création du jeu de données

Le jeu de données pour les tests est stocké dans un compte de stockage Azure, cela va nous permettre d'ajouter facilement de nouvelles images.

Accueil > ocwrk7336028223 >

 **azureml-filestore-81a6ebff-7004-4f0d-9981-894656...**
Partage de fichiers

»  Connecter  Charger  Ajouter un répertoire  Actualiser  Supprimer le partage

Nom	Type	Taille
 val	Répertoire	

Lors de la création de l'application ce répertoire sera monté dans la machine virtuelle.

Création de l'API

La base de l'API est créée sur le portail Azure.

Détails de l'instance

Vous avez besoin d'une base de données ? [Essayez la nouvelle expérience web + base de données.](#)

Nom * .azurewebsites.net

Publier * ☒ Code ☐ Conteneur Docker

Pile d'exécution *

Système d'exploitation * ☒ Linux ☐ Windows

Région *
 Vous ne trouvez pas votre plan App Service ? Essayez une autre région.

Plan App Service

Le niveau tarifaire du plan App Service détermine l'emplacement, les fonctionnalités, le coût et les ressources de calcul associés à votre application. [En savoir plus](#)

Plan Linux (Central US) *
 Créer

Référence et taille * **De base B1**
 100 ACU au total , 1.75 Go de mémoire

Pour l'accès au modèle ; on ajoute une variable d'environnement qui contient le mot de passe d'accès à la ressource. Cette variable d'environnement est **AZUREML_PASSWORD**

Nom

APPINSIGHTS_INSTRUMENTATIONKEY

APPLICATIONINSIGHTSAGENT_EXTENS

AZUREML_PASSWORD

Elle sera réutilisée dans le script :

```
svc_pr_password = os.environ.get("AZUREML_PASSWORD")
```

On ajoute le point de montage pour le jeu de données

Nom	Chemin de montage	Type	Nom du compte	Nom du partage
dataset	/dataset	AzureFiles	ocwrk7336028223	azureml-filestore-81a6eb

Développement de l'API

Les sources sont disponibles dans le répertoire
P8_07_Flask

L'application Flask affiche une page d'accueil permettant de saisir le n° de l'image ou d'en choisir une aléatoirement.

Le modèle est chargé au lancement de l'application.

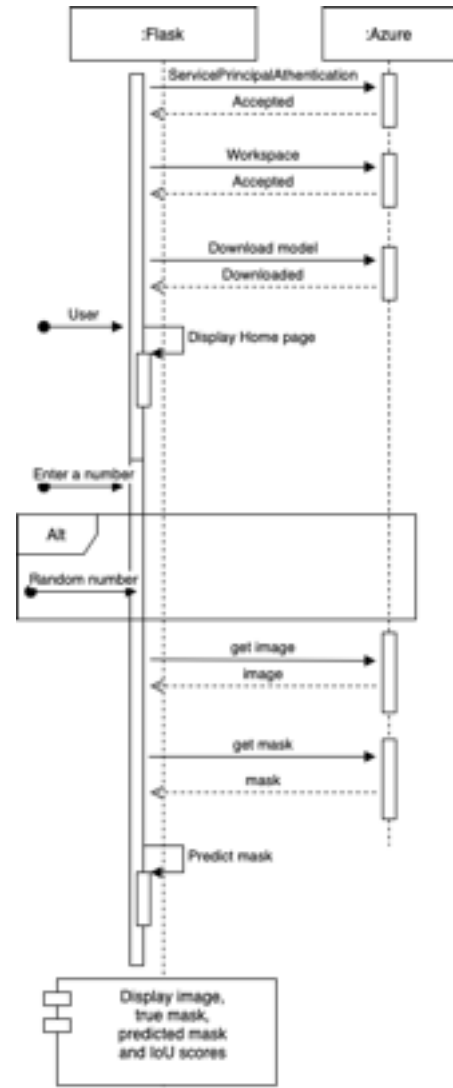
Lorsque l'utilisateur a saisi un nombre ou sélectionné un nombre aléatoire, l'application va chercher dans les répertoires Azure l'image et le masque correspondant à l'indice donné.

Le masque sera prédit par l'application Flask.

Une fois la prédiction faite les trois images (image d'origine, masque d'origine et masque prédit) sont affichées, ainsi que les scores IoU de chaque catégorie.


L'application est déployée sur Azure avec le script :

```
az webapp up --name fvt8
```



Rendu de l'API

<https://fvt8.azurewebsites.net>

**Future Vision Transport**






Image d'origine

2

Sélection d'une image dans le jeu de données

Lancer la prédiction

Au hasard Balthazar

Prédiction de la segmentation

Accuracy : 0.937%

Mean IoU : 0.295

Construction	0.473
Appartement	0.488
Humain	0.279
Nature	0.593
Objet	0.000
Ciel	0.000
Véhicule	0.440
Autre	0.495

Image annotée

Références

- **Fully Convolutional Neural Networks for Semantic Segmentation**
(<https://arxiv.org/abs/1411.4038>)
- **U-Net** <https://arxiv.org/pdf/1505.04597.pdf> (2015)
- **PSPNet**
<https://arxiv.org/pdf/1612.01105.pdf>, <https://hszhao.github.io/projects/pspnet/>
(2017)
- **CityScapes** <https://www.cityscapes-dataset.com>
- **ImageDataGenerator**
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- **Albumentations** <https://github.com/albumentations-team/albumentations>
- **Searching for MobileNetV3** <https://arxiv.org/pdf/1905.02244v5.pdf>
- **ServicePrincipalAuthentication** <https://github.com/Azure/MachineLearningNotebooks/blob/master/how-to-use-azureml/manage-azureml-service/authentication-in-azureml/authentication-in-azureml.ipynb>