Participez à la conception d’une voiture autonome

Note technique

Sommaire

1. **Introduction**
   1. Contexte du projet
   2. Etapes du projet
2. **Description et prétraitement des données**
   1. Description des données
   2. Prétraitement des données
3. **Segmentation sémantique des données**
   1. Fonctions de perte et métriques
   2. Modèles UNET
   3. Modèle VGG16-UNET
   4. Augmentation des données
4. **Comparaison des modèles**
5. **Déploiement du modèle optimal**
   1. API de prédiction
   2. Application Web
   3. Déploiement sur le cloud
6. **Conclusion et perspectives**
   1. Perspectives
   2. Conclusion
7. **Introduction**
   1. Contexte du projet

**Dans le cadre d’un projet de conception d’une voiture autonome, l’entreprise Future Vision Transport est sollicitée afin de réaliser un système embarqué.**

Ce système se divise en 4 parties :

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

En effet, la caméra embarquée de la voiture autonome capte des images qui doivent être traitées et segmentées avant de prendre une décision.

Les membre de mon équipe R&D traitent les autres parties du système embarqué et je traite la troisième partie c’est-à-dire la segmentation des images.

Il s’agit d’un problème de segmentation sémantique des images qui est une technique de Computer Vision.

Contrairement à la classification des images qui cherche à prédire à quelle catégorie appartient l’image, la segmentation sémantique des images cherche à prédire à quelle catégorie appartient chaque pixel de l’image.

* 1. Etapes du projet

Pour réaliser ce projet, on commence par télécharger les données sur le site « Cityscapes Dataset », ensuite on va entraîner des modèles Keras sur notre jeu de données. Afin d’améliorer le résultat, on va augmenter les données pour finalement choisir le modèle optimal qui sera mise mettre en production.

1. **Description et prétraitement des données**
   1. Description des données

Les données utilisées sont téléchargeables via ces deux liens [1](https://s3-eu-west-1.amazonaws.com/static.oc-static.com/prod/courses/files/AI+Engineer/Project+8+-+Participez+%C3%A0+la+conception+d'une+voiture+autonome/P8_Cityscapes_leftImg8bit_trainvaltest.zip), [2](https://s3-eu-west-1.amazonaws.com/static.oc-static.com/prod/courses/files/AI+Engineer/Project+8+-+Participez+%C3%A0+la+conception+d'une+voiture+autonome/P8_Cityscapes_gtFine_trainvaltest.zip)

On dispose des images des rue urbaines ainsi leurs masques. Ces images sont prises pendant la journée pendant plusieurs mois sur des périodes variées de l’année (printemps, été, automne). Les masques sont annotés selon 32 catégories :

Une image contenant texte

Description générée automatiquement

On va utiliser 2975 images pour entraîner les modèles, 250 images pour la validation et 150 images de test pour un contrôle final et pour comparer les modèles.

Exemple d’une image et son masque en couleur :

Une image contenant texte, bâtiment, extérieur, route

Description générée automatiquementUne image contenant texte, graphiques vectoriels

Description générée automatiquement

* 1. Prétraitement des données

Chaque images des données ont une taille de 1024\*2048\*3, on va redimensionner les images en 128\*256\*3 afin de réduire le temps de calcul.

Comme évoqué dans la section 2.1 les masques comportent 32 sous-catégories, mais on a besoin que de 8 catégories principales, on va commencer donc par transformer les sous-catégories en 8 catégories. Le but de l’algorithme ainsi est d’assigner à chaque pixel de l’image la valeur :

* 0 si le pixel est classifié comme **void**
* 1 si le pixel est classifié comme **flat**
* 2 si le pixel est classifié comme **construction**
* 3 si le pixel est classifié comme **object**
* 4 si le pixel est classifié comme **nature**
* 5 si le pixel est classifié comme **sky**
* 6 si le pixel est classifié comme **human**
* 7 si le pixel est classifié comme **vehicle**

1. **Segmentation sémantique des données**
   1. Fonctions de perte et métriques

Contrairement à la classification où l’entropie croisée qui est souvent utilisée pour optimiser les modèles de réseaux de neurones, la segmentation sémantique nécessite le test de plusieurs fonctions de perte afin de trouver le meilleur modèle. On testera dans ce projet :

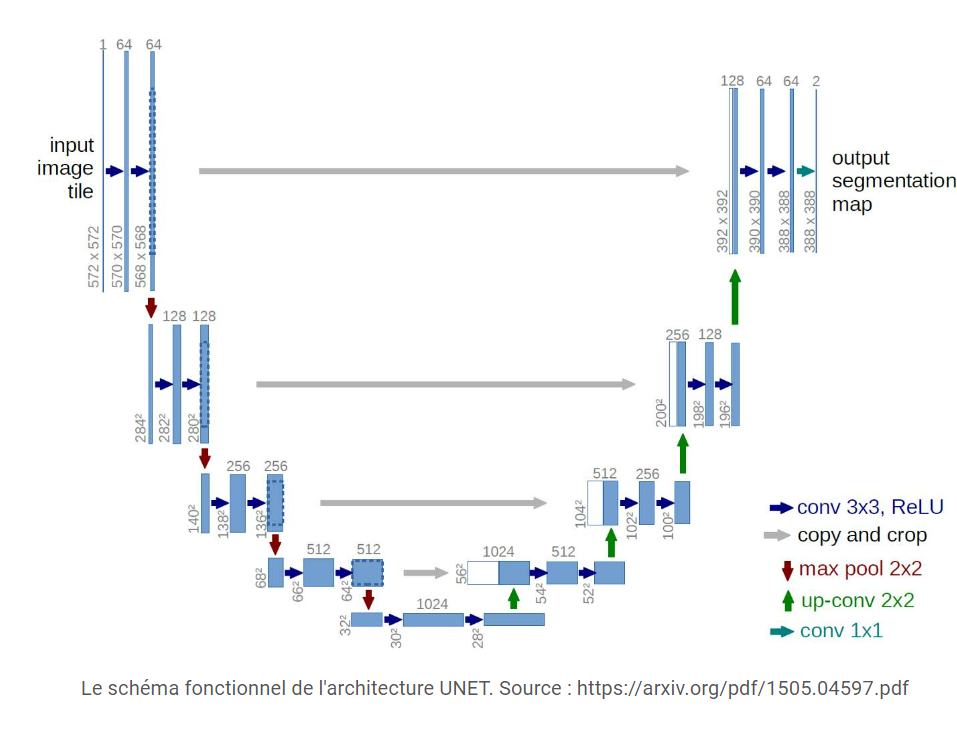
* L’entropie croisée avec logit
* L’entropie croisée équilibrée avec beta égale à 2
* La perte de Dice
* La perte totale

Comme on est dans un cas de classes déséquilibrées, l’accuracy n’est pas la métrique adaptée pour évaluer les métriques, c’est pour cela on va évaluer le modèle selon IoU (Intersection Over Union) et le coefficient de Dice.

* 1. Modèles U-NET

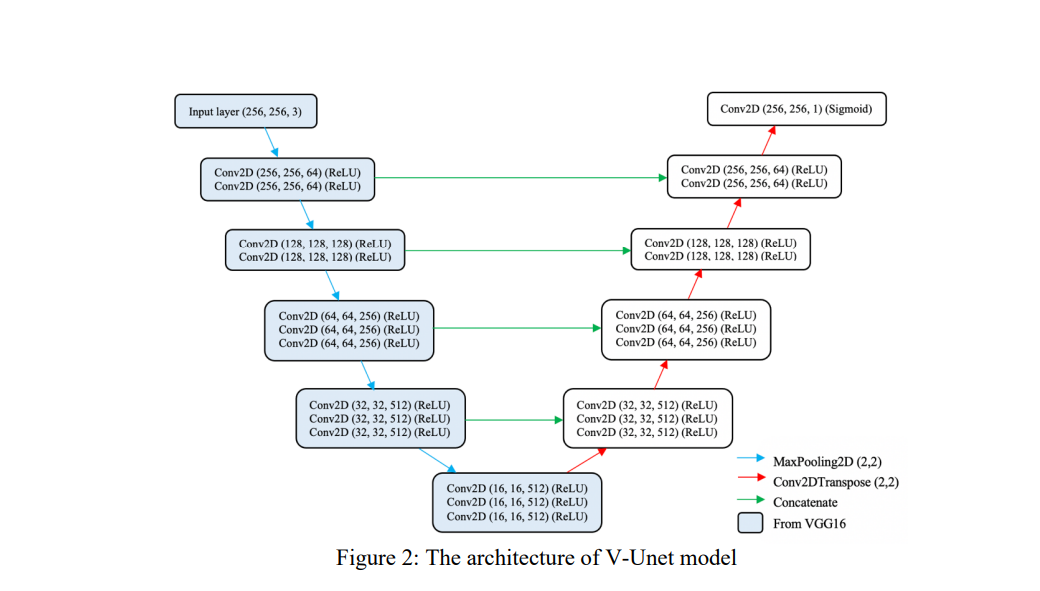
Le premier modèle qu’on va tester est un modèle de réseaux de neurones convolutifs qui était développé à la base pour segmenter des images biomédicales en 2015 à l'Université de Fribourg, en Allemagne. UNET est entièrement convolutif, la première partie du réseau est une série d’encodeurs et la deuxième se compose des décodeurs.

Ci-dessous l’architecture du modèle U-NET :



* 1. Modèle VGG16-UNET

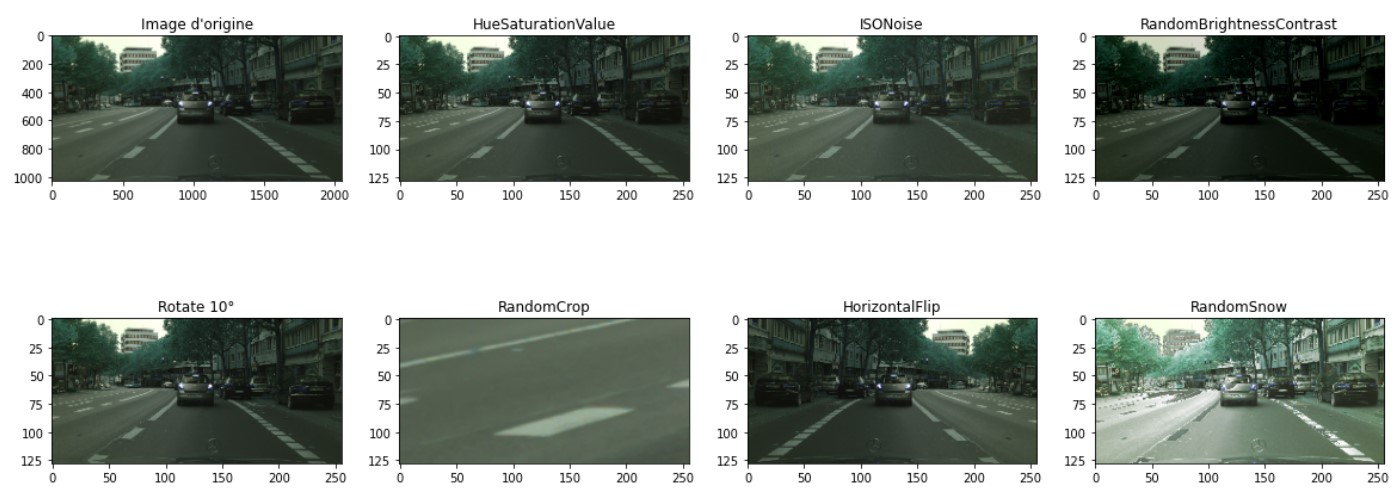
Comme on ne dispose pas d’assez de données pour entraîner le modèle UNET, utiliser un modèle déjà entraîné sur un grand dataset serait probablement une bonne idée pour améliorer la performance des réseaux UNET. Le VGG16-UNET est un bon candidat pour traiter le problème de segmentation sémantique. L’encodeur du VGG16-UNET est un réseau VGG16 pré entrainé entraîne sur les données d’ImageNet. Donc le modèle est capable de détecter les caractéristiques générique des images grâce au transfert learning.



* 1. Augmentation des données

Les modèles de Deep Learning tels que UNET et VGG16-UNET nécessite beaucoup de données pour les entraîner et éviter le surajustement. Cependant, il est souvent très difficile d'obtenir suffisamment d'échantillons d'apprentissage. Plusieurs raisons peuvent rendre très difficile, voire impossible, la collecte de suffisamment de données. Une façon de résoudre ce problème est la création des nouvelles images à partir des images existantes. On peut appliquer des légères modifications sur les images dont on dispose en faisant attention que les modifications apportées soient proches de la réalité. Il y a plusieurs bibliothèques pour augmenter les images comme « Albumentations » qu’on va l’utiliser dans la suite.

Dans notre cas, on a créé un générateur qui va augmenter les images et aussi paralléliser les calculs, c’est une classe MyGenerator qui hérite de la classe « Sequence » de Keras.utils ayant 3 méthodes : \_\_init\_\_, \_\_len\_\_ et \_\_getitem\_\_, ce générateur crée des nouvelles images en modifiant le contraste, la luminosité, en ajoutant un bruit ou/et en faisant une légère rotation ou/et en appliquant un zoom des images d’origine.



L’augmentation des données à amélioré les performances des modèles, on trouvera les détails dans le tableau récapitulatif dans la section suivante.

1. **Comparaison des modèles**

Ce tableau résume les résultats des modèles UNET et VGG-UNET entraînés sur des données avant et après l’augmentation des données

**Une image contenant table

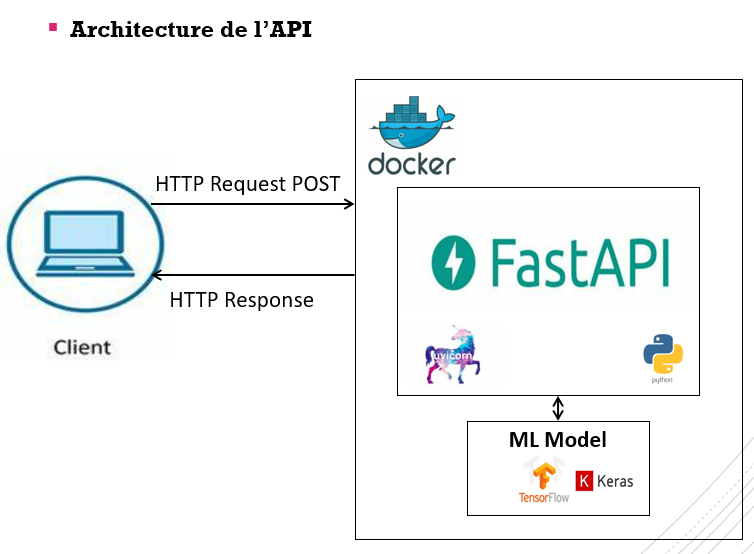
Description générée automatiquement**

Le meilleur modèle est le VGG-16 optimisé avec la perte de Dice et entraîné sur les données augmentées, ce modèle donne une IoU de 0.82 et accuracy de 0.89 et qui a nécessité environ 46 minutes pour l’entraînement, c’est ce modèle qu’on va mettre en production.

**Déploiement du modèle optimal**

* 1. API de prédiction

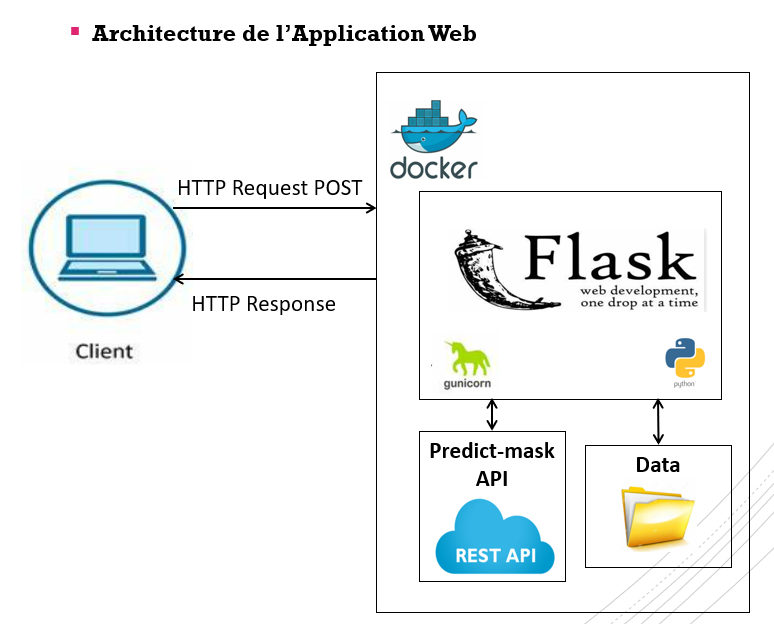
On a réalisé une API qui prend en entrée une image et renvoie le masque, ci-dessous l’architecture de l’API qui est réalisé avec FastApi. L’API a été conteneurisé avec Docker et envoyé vers Docker Hub. L’image est publique, elle peut être téléchargée en utilisant le nom de l’image **modestekhadija/predict\_mask-api**.Ci-dessous l’architecture de l’API :



* 1. Application Web

L’application Web est réalisée avec Flask, elle appelle l’API de prédiction et elle affiche l’image d’origine, le masque prédit et le masque réel.

L’application est conteneurisée avec docker et enregistrée avec le nom de l’image **modestekhadija/flask\_app-app**



* 1. Déploiement sur le cloud

L’API et l’application était déployer sur le Cloud avec Azure WebApp en utilisant les ressources gratuites d’Azure et sont exploitables via les URLs suivants :

* URL du déploiement de l’API :

<https://predict-mask.azurewebsites.net/>

* URL de déploiement de l’application Web :

<https://webapp-flask.azurewebsites.net/>

1. **Conclusion et perspectives**
   1. Perspectives

On a testé différents modèles : Unet et Vgg16. Afin d’améliorer les résultats, on peut envisager l’utilisation de modèles plus complexes qui sont entraînés sur un jeu de données similaire au notre comme DeepLab.

* 1. Conclusion

Ce projet constitue une introduction à la segmentation d’images qui est un domaine de la vision par ordinateurs. Notre mission dans ce projet est l’entrainement d’un modèle de Deep Learning afin de segmenter les images enregistrées par une caméra embarquée dans le cadre d’un projet de conception d’une voiture autonome. Le projet nous a permis :

* L’introduction à la segmentation d’images.
* L’introduction aux différents modèles de segmentation d’images.
* L’entrainement d’un large volume de data grâce au générateur d’images Keras.
* La réalisation d’une API Flask qui prédit le masque d’une image en entrée.
* La réalisation d’une API de présentation qui consomme l’API de prédiction.
* Le déploiement de l’API dans le cloud.