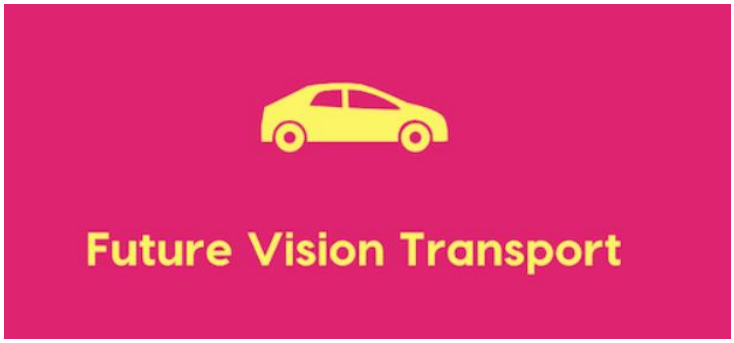


Conception et déploiement d'un modèle de segmentation d'images

- - - - -

Note technique

SOMMAIRE

A - INTRODUCTION

1 - Présentation du projet

Ce document a pour objectif de présenter le travail effectué et les résultats obtenus dans le cadre du projet Future Vision Transport.

Future Vision Transport est une entreprise qui conçoit des systèmes embarqués de Vision par ordinateur pour les véhicules autonomes. Chaque système se compose de 4 parties :

- Partie 1 : acquisition des images en temps réel,
- Partie 2 : traitement des images,
- Partie 3 : segmentation des images,
- Partie 4 : système de décision,

⇒ Le but du projet est de travailler sur la partie 3 : **la segmentation d'images**.

2 - Objectif du projet

L'objectif de ce projet est donc de concevoir un modèle de segmentation d'images. Ce modèle sera alimenté par le bloc de traitement des images et il alimentera lui-même le système de décision.

3 - Contraintes à respecter

Les contraintes à respecter pour ce projet sont les suivantes :

Données en entrée :

- Le jeu de données utilisé est **Cityscape** et il faut travailler avec les 8 catégories principales.
- Les images en entrée peuvent changer.
- Le volume des données est important.

Résultat :

- Le résultat de la segmentation doit être rendu dans une API Flask simple d'utilisation et qui sera déployée sur le Cloud.
- Une application Flask de présentation des résultats qui consomme l'API de prédiction, déployée également sur le Cloud. Cette application sera l'interface pour tester l'API et intégrera les fonctionnalités suivantes : affichage de la liste des ID des images disponibles, lancement de la prédiction du mask pour l'ID sélectionné par appel à l'API, et affichage de l'image réelle, du mask réel et du mask prédit.

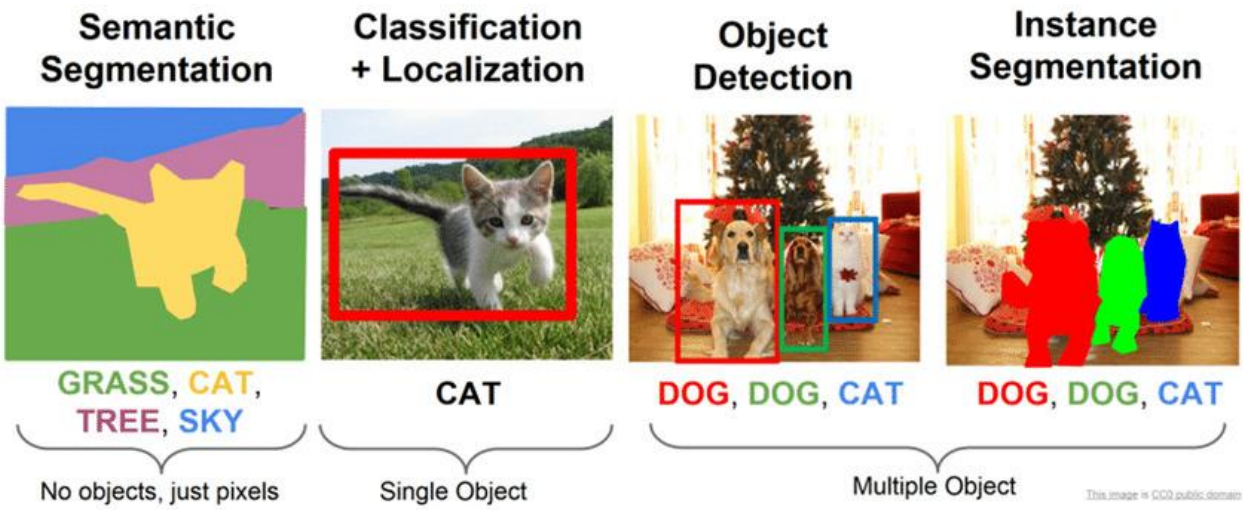
B - PRÉSENTATION DE L'ÉTAT DE L'ART DE LA SEGMENTATION D'IMAGES

1 - Vision par ordinateur

La segmentation d'images fait partie du domaine de l'intelligence artificielle appelé Vision par Ordinateur "Computer Vision". Ce dernier comporte trois principales tâches :

- La Classification : consiste à classifier des images selon des catégories préalablement établies : l'image d'un chien ou d'un chat. Chaque image ne peut appartenir qu'à une seule catégorie.
- La Détection d'objets : consiste à identifier pour tous les objets présents sur l'image leur position et leur catégorie. Une image peut comporter plusieurs catégories.
- La Segmentation : consiste à classer chaque pixel d'une image selon sa catégorie.

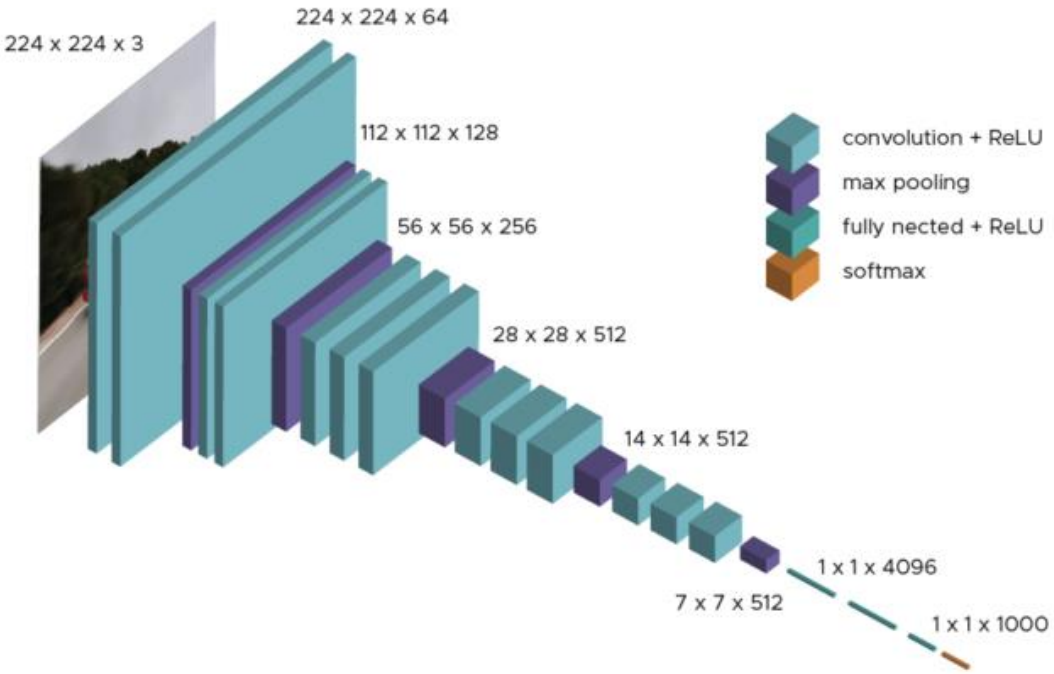
La segmentation d'images qui nous intéresse utilise des techniques de classification. Nous allons donc décrire les algorithmes utilisés pour cette tâche avant de décrire en détails les algorithmes pour la segmentation.



Principales tâches de la vision par ordinateur

2 - Classification d'images

La classification utilise des réseaux de neurones à convolution classiques (CNN): ces réseaux sont un enchaînement de couches de convolution et de Pooling terminées par une ou plusieurs couches denses (Fully Connected). Les couches de convolution servent à extraire les caractéristiques des images (Features) et produisent en sortie des cartes de caractéristiques (Features maps). Ces cartes de caractéristiques contiennent les éléments (pattern) identifiés par le réseau dans l'image. Les couches de Pooling permettent de résumer l'information recueillie par les cartes de caractéristiques et servent à diminuer le nombre de paramètres du réseau. Le Pooling le plus utilisé est le **Max Pooling**. Les dernières couches denses permettent de faire la classification et se terminent par une couche avec une fonction d'activation de type Sigmoid (Logistique si N = 2 et et Softmax si N > 2). Un des réseaux de ce type les plus connus est **VGG Net**.



Représentation 3D de l'architecture de VGG-16

VGG-16 est constitué de plusieurs couches, dont 13 couches de convolution et 3 fully-connected. Il doit donc apprendre les poids de 16 couches.

Il prend en entrée une image en couleurs de taille 224 × 224 px et la classifie dans une des 1000

classes. Il renvoie donc un vecteur de taille 1000, qui contient les probabilités d'appartenance à chacune des classes.

Plusieurs évolutions ont permis au cours du temps d'améliorer ces réseaux. Voici les principales :

- Réseaux FCN (Fully Convolutional Network) : on remplace les couches finales fully connected par une ou plusieurs couches de convolution. Le réseau FCN n'a donc pas de couche fully connected.
- Réseaux ResNet (Residual Network, développés par Microsoft) : on ajoute des blocs résiduels (skip connections) qui créent une connexion entre la sortie d'une couche de convolution et leur entrée originale, ce qui permet de ne pas oublier l'information initiale.
- Réseaux Inception (développés par Google) : on ajoute des blocs inception qui sont des couches de convolution parallèles et concaténées (concept de réseau dans le réseau (Network In Network)). Ces couches améliorent les performances en augmentant la non linéarité et en permettant de faire une analyse à plusieurs échelles spatiales grâce aux branches en parallèle qui sont des couches de convolutions avec des filtres de tailles différentes.

Ces algorithmes de classification seront utilisés pour la segmentation en tant que "Backbone". C'est-à-dire qu'ils seront utilisés pour extraire les caractéristiques des images.

3 - Segmentation sémantique

La segmentation sémantique est une tâche de classification qui consiste à attribuer une classe pour chaque pixel de l'image.

Pour réaliser cette tâche, on utilise des algorithmes basés sur la technologie Encoder-Decoder: l'Encoder diminue la dimension de l'image (downsampling), puis le Decoder la réaugmente (upsampling). L'Encoder diminue la dimension de l'image en extrayant les caractéristiques de l'image grâce à un réseau convolutif utilisé pour la classification (Backbone). Le Decoder augmente la dimension afin de retrouver la dimension de l'image originale et créer une image de sortie (appelée masque) contenant la segmentation, c'est-à-dire la catégorie correspondant à chaque pixel.

Cet upsampling peut se faire de deux façons :

- Approche sans paramètres : plus proches voisins (nearest neighbours), max unpooling
- Approche avec paramètres : on utilise des filtres appris par le réseau. ¶

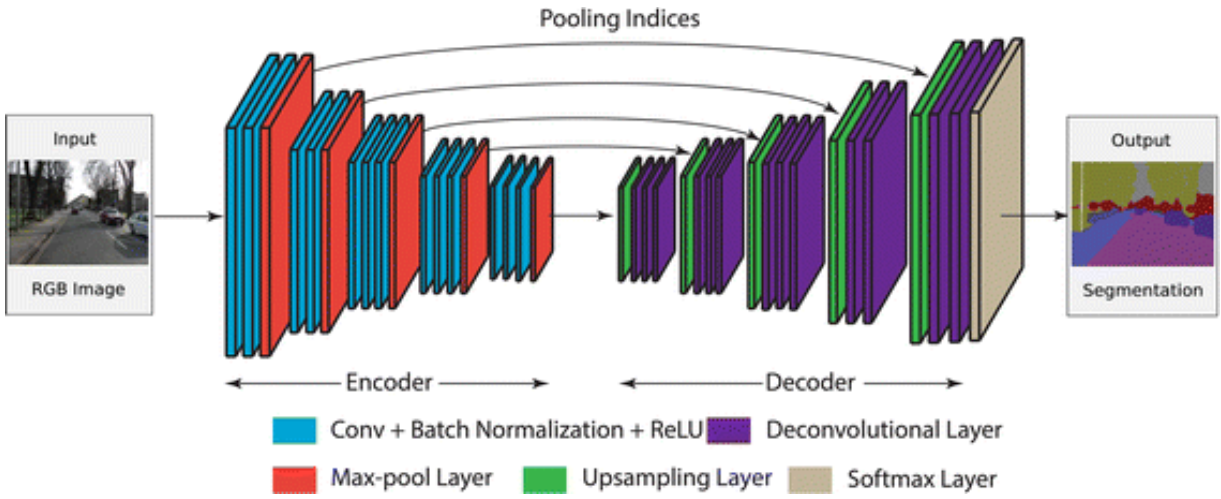


Schéma de l'architecture Encoder-Decoder

Le problème avec ce type d'architecture simple est une perte d'information spatiale fine : l'approche sablier fait perdre l'information de haute résolution qui est nécessaire pour la segmentation. Pour corriger cela, on ajoute des connexions entre les couches de même résolution (skip connections). Pour réaliser ces connexions, on prend le tenseur de la phase de downsampling et on vient le concaténer lors de la phase d'upsampling. Cette technique est très efficace car elle ne rajoute pas de paramètres.
L'architecture emblématique de ce type de réseau est Unet qui doit son nom à sa forme typique en U.

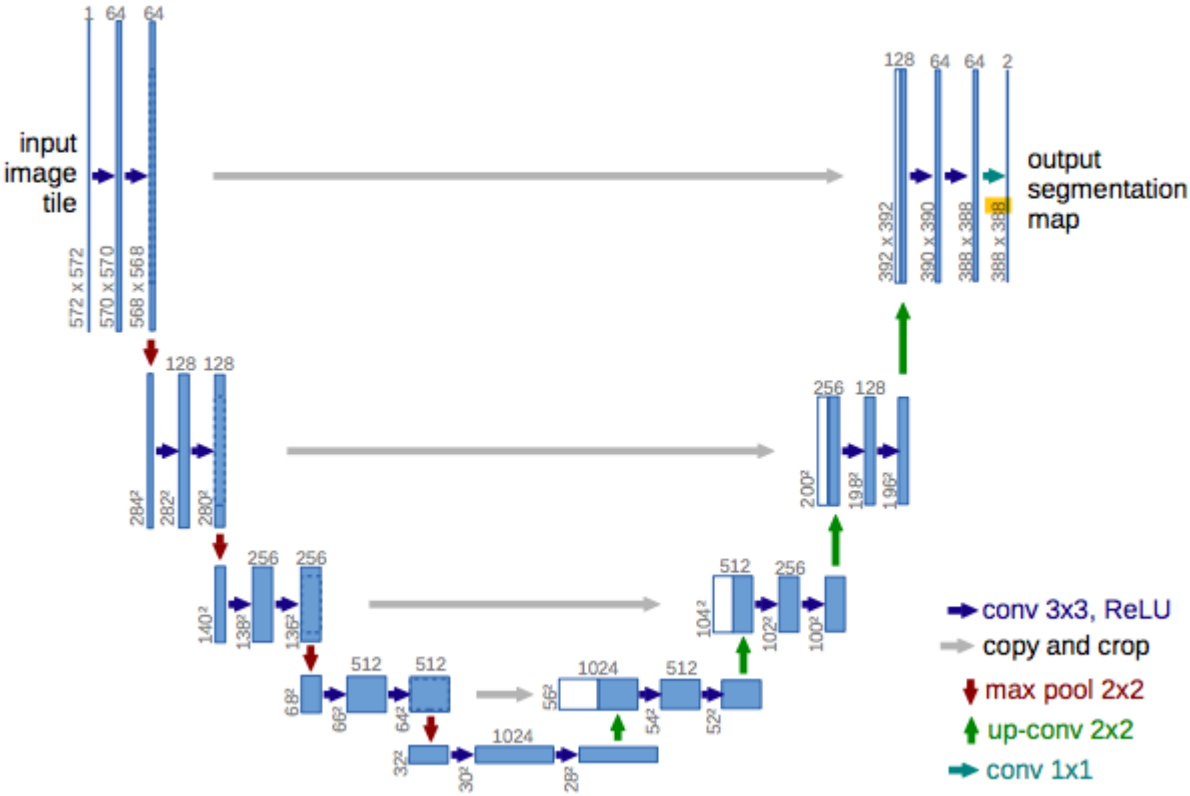


Schéma de l'architecture Unet

Cette architecture permet une d'extraire des caractéristiques et corrige la baisse de résolution engendrée grâce à des connexions entre couches. D'autres architectures peuvent être utilisées comme Linknet, PSPNet ou FPN. Ces architectures sont basées sur le même principe que Unet mais avec des modifications, notamment au niveau des connexions entre couches.

C - PRÉPARATION DE LA MODÉLISATION

1 - Présentation des données

Nous utilisons pour ce projet les données issues du dataset Cityscapes. Ce jeu de données comporte des images qui sont des photos prises d'une voiture dans différentes villes. Le dataset contient également les annotations, appelées masques, de chaque image. Le masque est également une image dont les pixels correspondent à la catégorie dans l'image originale (piéton, nature, véhicule...).

Exemples issus du jeu de données Cityscape.

2 - Préparation des données

a - Séparation du jeu de données

Le dataset contient les jeux de données suivants : Train (2975 images et masques), Validation (500 images et masques) et Test. Les masques pour le jeu de Test ne sont pas disponibles, nous ne pourrons donc pas les utiliser pour évaluer le modèle final.

Pour remédier à ce problème nous avons procédé à la séparation du jeu de données de la façon

suivante :

- Nous avons créé un jeu de données Test à partir du jeu de données Validation original. Ce jeu de données Test nous permettra d'évaluer le meilleur modèle final optimisé.

- Nous avons séparé le jeu de données Train original en un jeu de données Train et un jeu de données Validation d'après un ratio TRAIN_VAL_SPLIT spécifié.

Le jeu de données Train nous permettra d'entraîner les différents modèles.

Le jeu de données Validation nous permettra d'évaluer les différents modèles, de les comparer et de les optimiser.

- Le jeu de données Test original n'est pas utilisé.

- Nous avons également créé des jeux de données 'Train échantillon' et 'Validation échantillon' afin d'entraîner les modèles intermédiaires sur un petit échantillon d'images afin de faire des tests.

Nous avons été très vigilant à ne pas toucher au jeu de données Test afin qu'il n'y ait pas de fuite de données.

b - Mise en place des 8 catégories principales

Le jeu de données Cityscape contient 8 catégories et 32 sous-catégories. Comme cela nous est demandé nous allons travailler sur les 8 catégories essentielles.

Les 8 catégories sont : **void**, **flat**, **construction**, **object**, **nature**, **sky**, **human** et **vehicle**. Pour cela, nous avons implémenté un dictionnaire et une fonction de mapping permettant de passer des 32 sous-catégories aux 8 catégories.

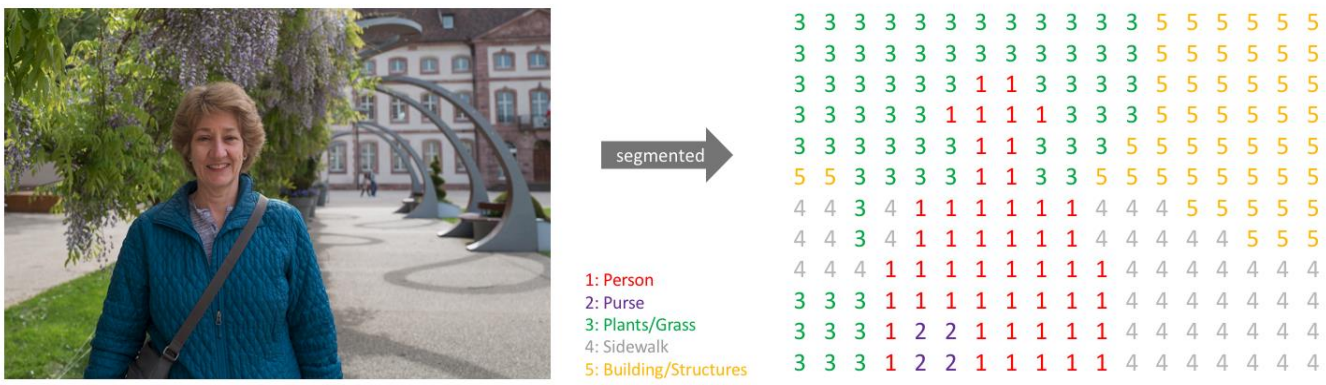
c - Renommage des images et masques

Afin de respecter la contrainte d'utiliser des identifiants (ids) pour les images et masques, nous les avons renommés à l'aide d'identifiants (entiers numériques). Cela permet de manipuler plus facile les images et les masques.

3 - Identification de la cible

L'objectif du projet est de faire de la segmentation d'images, c'est à dire de réaliser un découpage de chaque image originale en différentes catégories prédéfinies. Pour cela il faut prédire à quelle catégorie appartient chaque pixel de l'image original. L'ensemble des prédictions de l'appartenance des pixels aux catégories va former une image que l'on appelle un masque.

La cible à prédire (également appelé le Label') est donc le masque de chaque image qui correspond à la catégorie de chaque pixel de l'image. Pour chaque image en entrée, il faut donc prédire un masque contenant pour chaque pixel la catégorie correspondante à ce pixel (nature, ciel, humain…).



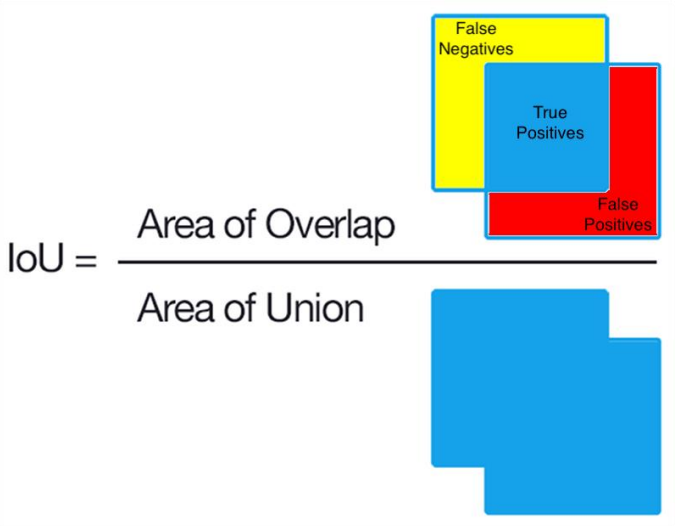
Exemple de segmentation d'une image: résultat sous forme d'une matrice contenant les catégories de chaque pixel.

D - MODÉLISATION

1 - Choix de la métrique d'évaluation

La métrique d'évaluation classique utilisée dans les problèmes de classification est l'Accuracy. Cependant cette métrique n'est pas adaptée à des jeux de données déséquilibrés comme c'est le cas pour la segmentation d'images.

Nous avons retenu la métrique **Mean IoU Score** pour évaluer les modèles. Ce score correspond à la moyenne du coefficient "**Intersection over Union, IoU**" des différentes catégories. Le coefficient IoU est défini comme la zone de chevauchement (overlap) entre la segmentation prédite et la segmentation réelle divisée par la zone d'union entre la segmentation prédite et la segmentation réelle.



Représentation graphique du coefficient "Intersection over Union".

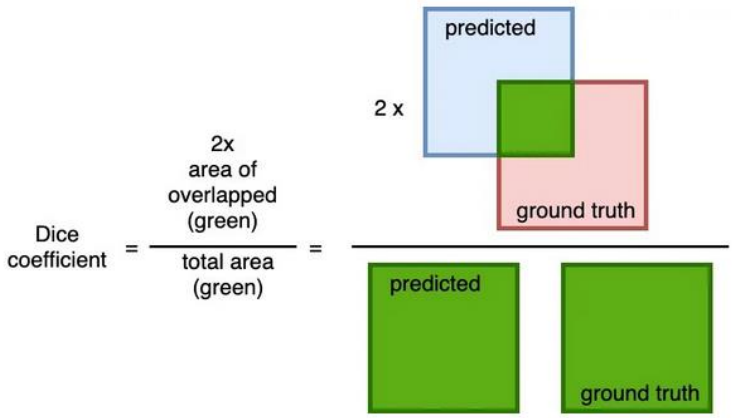
La métrique Mean IoU Score est particulièrement bien adaptée à la problématique métier de segmentation d'images. Le score est compris entre 0 et 1 et plus on s'approche de 1 plus la similarité entre la prédiction et la réalité est élevée et meilleure est la segmentation.

En plus de cette mesure, nous avons également pris en compte le temps d'exécution et le nombre de paramètres des différents modèles.

2 - Fonction de perte

La fonction de perte classique pour les modèles de classification à plusieurs catégories est la "Categorical Cross Entropy". Lors de la phase d'optimisation des hyperparamètres, nous chercherons à optimiser cette fonction. Pour cela nous essaierons une fonction plus adaptée à la segmentation d'images : la fonction de perte "**Dice Loss**".

La fonction de perte Dice Loss est calculée à partir du coefficient Dice qui est égal à deux fois la zone de chevauchement entre la prédiction et la réalité divisé par le nombre total de pixels de l'image et du masque.



Représentation graphique du coefficient de Dice.

3 - Manipulation d'un jeu de données volumineux

Les images sont des données de grande dimension et sont plus volumineuses et plus difficiles à gérer que des données tabulaires classiques. Le volume important des images fait que le jeu de données ne peut pas être entièrement chargé en mémoire. Pour pallier à ce problème, nous avons mis en place un générateur de données.

Le générateur de données est implémenté sous forme de classe Python qui “dérive” de la classe Keras “Sequence”. Le générateur de données fournira les images au modèle au fur et à mesure de l'apprentissage sous forme de batchs. La taille du batch correspond au nombre d'images envoyées en même temps au modèle. Le générateur de données permet également le traitement des images sur plusieurs cœurs de calcul.

4 - Augmentation des données

Le nombre d'images annotées est relativement faible car l'annotation des images requiert un travail important. C'est un problème classique que l'on retrouve en segmentation d'images. Pour pallier à ce ce problème, nous avons utilisé la technique d’augmentation des données. Pour ce faire, nous avons utilisé la librairie Albumentation. Cette dernière permet d'augmenter les images en appliquant des transformations aux images originales (rotation, ajout de bruit gaussien, etc.). Cela permet donc d'obtenir de nouvelles images à partir des images originales et in fine d'améliorer l'apprentissage. L’augmentation des données se fait à la volée dans le générateur de données.

Nous avons testé deux types d'augmentation d'images :

- Une première version “Soft” qui utilise trois techniques d'augmentation :
  - 1 : Horizontal Flip (basculement horizontal de l'image)
  - 2 : Random Gamma (modification aléatoire du gamma)
  - 3 : Blur (ajout de bruit)

Chacune des trois techniques d'augmentation a une probabilité de 0.25 d'être appliquée

- Une seconde version plus “Hard” qui utilise six techniques d'augmentation :
  - 1 : Elastic Transform (transformation élastique)
  - 2 : Horizontal Flip (basculement horizontal)
  - 3 : Grid Distortion (distorsion de l'image)
  - 4 : Random Gamma (modification aléatoire du gamma)
  - 5 : Emboss (ciselage de l'image)
  - 6 : Blur (ajout de bruit)

Chacune des six techniques d'augmentation a une probabilité de 0.5 d'être appliquée.¶

La seconde version transforme plus les images que la première : le nombre de modifications et la probabilité d'application sont plus élevés.

Augmentation de données – version SOFT: image originale à gauche, image transformée à droite.

Augmentation de données – version HARD: image originale à gauche, image transformée à droite.

5 - Implémentation des modèles et optimisation des hyper paramètres

Les modélisations que nous avons implémentées peuvent être regroupées en 2 catégories :

- *Modélisations "from scratch"* : les modèles sont implémentés 'à la main' en utilisant Keras. Cette première approche permet de bien comprendre et appréhender les modèles et l'API Keras.
- *Modélisations utilisant la librairie "segmentation-models"* : cette librairie Python, qui s'appuie sur Keras, contient des modèles, des métriques et des fonctions de coût dédiés à la segmentation.

Cette seconde approche permet d'utiliser des modèles plus avancés, ainsi que des fonctions de coût différentes déjà implémentées.

L'utilisation de cette librairie nous permettra notamment d'optimiser les hyperparamètres :

✓ La fonction de coût : essai de la fonction Dice Loss en plus de la fonction Categorical Cross Entropy.

✓ Le Backbone : essais des backbones VGGNet, ResNet et EfficientNet avec et sans poids préentrainés sur les données Imagenet pour l'initialisation.

✓ L'architecture : essais des architectures Unet, FPN (Features Pyramid Network) et Linknet.

Les différentes modélisations implémentées forment une suite logique dont le but est d'améliorer le modèle au fur et à mesure afin d'obtenir le meilleur modèle qui sera déployé.

Pour l'entraînement des modèles nous avons réduit la taille des images afin d'avoir des temps de calcul raisonnables et également pour tenir compte de la capacité mémoire du matériel qui nous est alloué.

6 - Tableau comparative des résultats de différents modèles

Modèle de référence
Meilleur modèle
Meilleur modèle entraîné sur des images de plus grande taille

	Caractéristiques¶				
N°	**Implementation**	**Model**	**Backbone**	**Loss Function**	**Augmentation**
1	*from scratch*	*Unet*	no	*Categorical Cross Entropy*	*no*
2	from scratch	Unet	no	Categorical Cross Entropy	Soft
3	from scratch	Unet	no	Categorical Cross Entropy	Hard
4	segmentation-models	Unet	VGGnet	Categorical Cross Entropy	Soft
5	segmentation-models	Unet	VGGnet - Imagenet	Categorical Cross Entropy	Soft
6	segmentation-models	Unet	VGGnet - Imagenet	Dice Loss	Soft
7	segmentation-models	Unet	Resnet - Imagenet	Dice Loss	Soft
8	segmentation-models	Unet	Efficientnet - Imagenet	Dice Loss	Soft
9	segmentation-models	FPN	Efficientnet - Imagenet	Dice Loss	Soft
10	segmentation-models	Linknet	Efficientnet - Imagenet	Dice Loss	Soft
11	**segmentation-models**	**FPN**	**Efficientnet - Imagenet**	**Dice Loss**	**Soft**

	RÉSULTATS					
N°	**Nb params**	**Exec Time**	**Loss**	**Val_Loss**	**Mean IoU**	**Val_Mean IoU**
1	*31 059 592*	*2h41mns*	*0.2608*	*0.3565*	*0.6478*	*0.5968*
2	31 059 592	2h27mns	0.3177	0.3118	0.5972	0.5719
3	31 059 592	3h16mns	0.6106	0.5002	0.3996	0.4750
4	23 753 288	2h20mns	0.3862	0.4546	0.5423	0.4771
5	23 753 288	1h58mns	0.3319	0.3099	0.5858	0.5871
6	23 753 288	2h18mns	0.1917	0.1959	0.7023	0.7010
7	24 457 169	1h55mns	0.1856	0.2150	0.7123	0.6812
8	17 868 848	2h48mns	0.1605	0.1746	0.7446	0.7299
9	13 919 792	2h08mns	0.1441	0.1713	0.7659	0.7332
10	13 762 000	2h23mns	0.1629	0.1803	0.7422	0.7236
11	**13 919 792**	**2h17mns**	**0.1311**	**0.1474**	**0.7845**	**0.7639**

Analyse des résultats :

Le premier modèle est le modèle Unet sans augmentation des données et avec des hyperparamètres standards. Ce modèle nous servira de référence (baseline). Ce modèle donne un score IoU sur le jeu de validation de 0.59.

Augmentation des données : la version SOFT de l'augmentation des données améliore les résultats en termes d'apprentissage alors que la version HARD les dégrade complètement. On peut en déduire

qu'une transformation modérée des images est bénéfique pour l'apprentissage mais qu'une transformation trop importante lui est défavorable. Cela peut s'expliquer par le fait que la version SOFT effectue une transformation de l'image mais tout en gardant sa structure, alors que la version HARD destructure l'image comme nous l'avons vu dans l'exemple précédent.

Fonction de coût : la fonction de coût "Dice Loss" améliore les résultats par rapport à la fonction Categorical Cross Entropy. Cela confirme que la fonction de coût "Dice Loss" est bien adaptée à la problématique de la Segmentation.

Backbone : le backbone qui donne les meilleurs résultats est un réseau EfficientNet avec les poids pré entraînés sur le jeu de données Imagenet. L'utilisation de poids pré entraînés améliore les résultats, ainsi que le réseau EfficientNet qui augmente le score IoU tout en diminuant le nombre de paramètres et le temps d'exécution.

Architecture : l'architecture qui donne les meilleurs résultats est FPN. Cette architecture améliore nettement les résultats : le score IoU augmente et le temps d'exécution diminue.

Le meilleur modèle est donc le suivant :

- Augmentation des données version SOFT.
- Fonction de coût : Dice Loss.
- Backbone : EfficientNet avec poids pré entraînés sur Imagenet.
- Architecture : FPN.

Grâce à ce modèle nous obtenons de meilleurs résultats par rapport au modèle Baseline :

- Le score IoU de validation augmente nettement de 0. 59 à 0.73.
- Le temps d'exécution baisse de 2h41mns et 2h08mns.
- Le nombre de paramètres reste très raisonnable (13 919 792).

Nous avons ensuite entraîné ce meilleur modèle sur des images de plus grande taille. Cela a encore nettement amélioré le résultat : nous obtenons un score IoU de validation égal à 0.76, ce qui est un très bon score.

Evaluation du meilleur modèle sur le jeu de test :

Pour finir, nous avons évalué le meilleur modèle sur le jeu de données de test :

- On obtient un score IoU sur le jeu de test égal à 0.762.
- Ce score est très proche de celui obtenu lors de la modélisation sur le jeu de validation.
- Cela montre la qualité et la bonne capacité de généralisation de notre modèle.

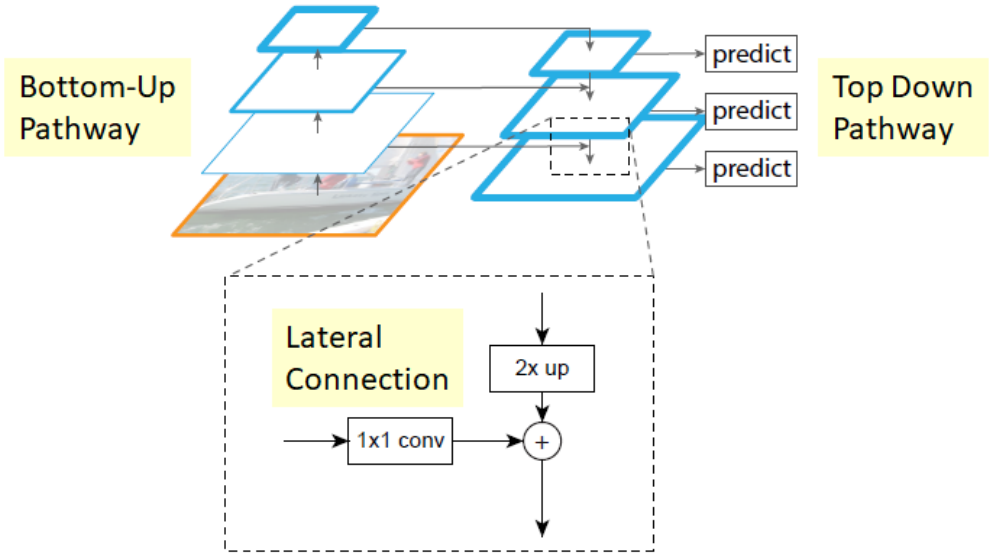
7 - Présentation détaillée du meilleur modèle

Backbone EfficientNet :

Les concepteurs de ce réseau ont constaté qu'habituellement on a tendance à augmenter les réseaux de neurones, en termes de largeur, profondeur et résolution, au fur et à mesure que la puissance de calcul augmente. Mais cette augmentation se fait de façon assez empirique voire aléatoire. L'idée des concepteurs est de rationaliser le scaling (mise à l'échelle). Pour cela, dans un réseau EfficientNet, le scaling se fait de façon intelligente grâce à une formule mathématique qui permet d'ajuster la largeur, la profondeur et la résolution de façon homogène et proportionnelle. Cela permet d'obtenir des réseaux optimisés en termes de dimensions et par conséquent de nombre de paramètres.

Architecture FPN (Features Pyramid Network) :

Les réseaux FPN se composent de deux phases : une première phase appelée 'bottom-up' et une seconde phase appelée 'top-down'. Lors de la phase bottom-up, on construit une pyramide de cartes de caractéristiques. Lors de la phase top-down, on utilise les cartes de caractéristiques en pyramide de la précédente phase pour enrichir celles de la phase top-down lors de l'upsampling. Pour cela, les cartes de caractéristiques des 2 phases du même niveau, c'est-à-dire ayant la même résolution, sont regroupées en faisant une addition termes à termes. On a donc des connexions latérales à chaque étage de la pyramide des cartes de caractéristiques de différentes résolutions, ce qui permet d'enrichir l'information lors de la partie upsampling.



Schémas de l'architecture FPN. Les cartes de caractéristiques de la phase 'bottom-up' subissent des convolutions 1×1 pour réduire les dimensions des canaux.

Le meilleur modèle est enregistré au format H5 afin d'être déployé via une API REST. ¶

E - API REST

1 - Implémentation d'une API et d'une application Web

Nous avons déployé le meilleur modèle via une API en utilisant le framework Flask.

https://api-prediction-7ocqxqp7xa-ew.a.run.app

Le déploiement s'est fait sur Google Cloud Run. Cette API est ensuite consommée par une application Web en utilisant le framework Flask.

Dans cette application Flask nous avons créé 2 routes (endpoints) :

- une première route qui retourne la page d'accueil, celle-ci contient :
  - un titre "Segmentation d'image",
  - la liste des identifiants des images disponibles pour la segmentation : cela permet à l'utilisateur de savoir rapidement quelles sont les images disponibles et prêtes pour une segmentation. Cette page affiche automatiquement le couple (image, mask) correspondant à l'ID choisi.
- - une deuxième route qui retourne le résultat de la segmentation : cette route prend en entrée l'identifiant d'une image fournie par l'utilisateur dans sa requête et retourne l'image avec les segments identifiés par le modèle, via l'API précédente, et une superposition de ce Mask avec l'image qui a été choisie par l'utilisateur.

2 - Déploiement de l'API et de l'application Web

Ensuite nous avons déployé ces applications. Pour cela nous avons utilisé le service de Google Cloud Run. Chaque application est implémentée dans les fichier "main.py".

- Configuration de Google Cloud :
  - Créer un nouveau projet
  - Activez l'API Cloud Run et l'API Cloud Build
- Installer et initier le SDK Google Cloud : https://cloud.google.com/sdk/docs/install
- Créer les fichiers "Dockerfile", "requirements.txt", et ".dockerignore" :
  - https://cloud.google.com/run/docs/quickstarts/build-and-deploy#containerizing
- Construction et déploiement dans le Cloud :
  - gcloud builds submit --tag gcr.io/<project_id>/<function_name>
  - gcloud run deploy --image gcr.io/<project_id>/<function_name> --platform managed

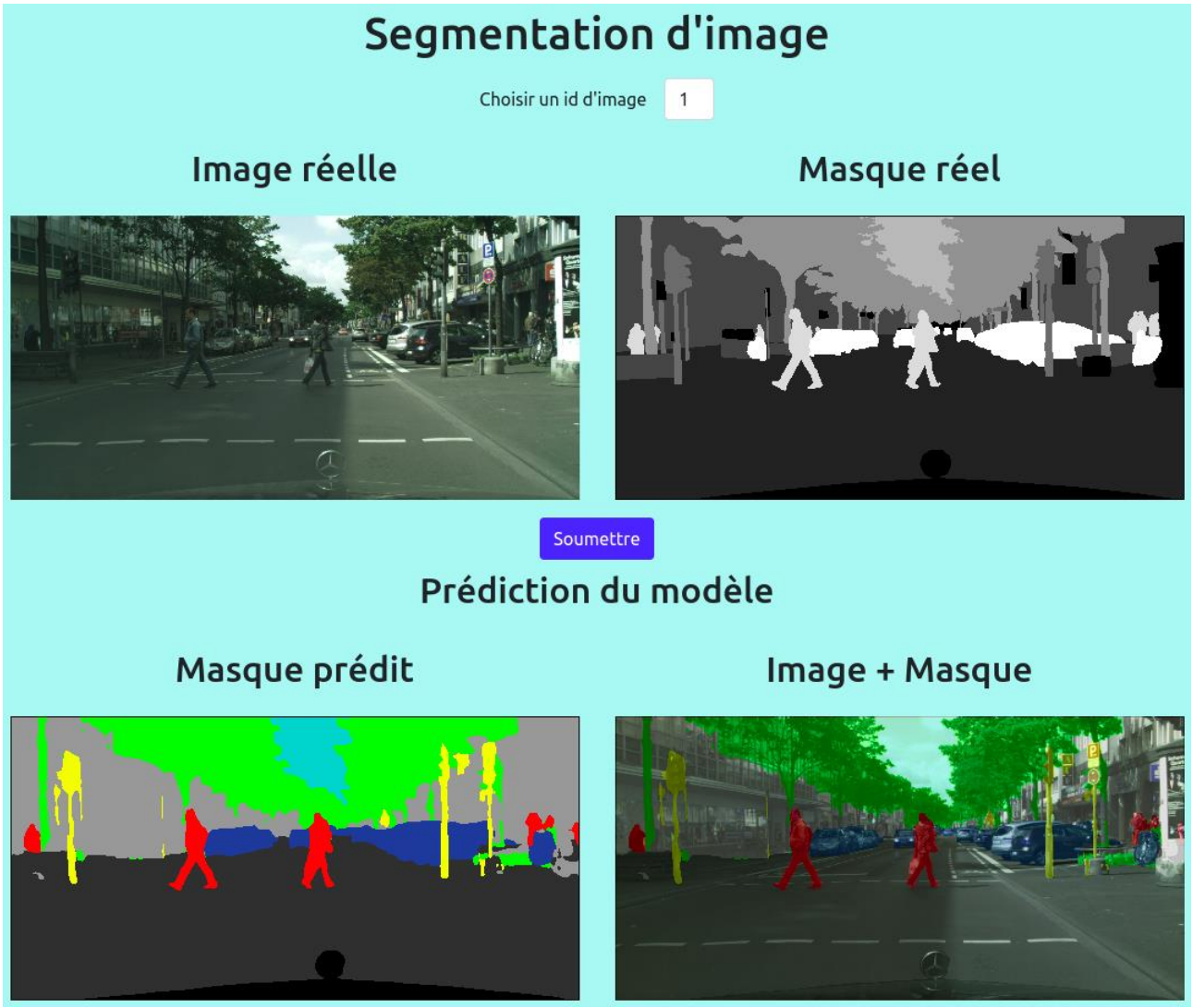
➢ **adresse de l'application déployée sur Google Cloud via Cloud Run** :

https://segment-uwljdhbvxa-ew.a.run.app/

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*”.

➢ **Résultat obtenu** : exemple de résultat renvoyé par l’API pour une segmentation d’image :



F - CONCLUSION ET PISTES D'AMÉLIORATIONS

L’objectif fixé a été atteint : nous obtenons un modèle de qualité qui donne de bons résultats et avec une bonne capacité de généralisation, tout en ayant respecté les contraintes imposées.

Voici quelques pistes d’améliorations possibles : on pourrait essayer :

- d’optimiser d’autres hyperparamètres comme l’optimiser (SGD, RMSprop, dagrad…).
- d’autres versions d’augmentation des images.
- d’entraîner les modèles sur des images de plus grande taille nécessiterait plus de ressources de calcul.
- d’autres fonctions de perte, par exemple une combinaison linéaire entre la Categorical Cross Entropy et la Dice Loss.
- une architecture de type Mask R-CNN : architecture basée sur les modèles de détection tel que Faster R-CNN.
- Le déploiement continu qui pourrait prendre en compte plus facilement ou automatiquement un changement de version de l’application.