Segmentation sémantique avec Cityscapes

Sylvia BANKOWSKA - Novembre 2022

Le présent article expose les étapes qui ont mené à la construction du modèle de segmentation sémantique déployé dans le cadre du projet lancé par la société Futur Vision Transport visant à concevoir la voiture autonome.

Table des matières

1. Enjeux et objectifs

Le projet de conception d'une voiture autonome s'inscrit dans le cadre de la construction de systèmes embarqués de **computer vision** (vision par ordinateur) pour les véhicules, menée par la société *Future Vision Transport*.

La chaîne de traitement d'un tel système se compose de plusieurs parties :

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

Nous avons été sollicités pour la 3ème partie : la segmentation des images.

Du point de vue entreprise, notre rôle était de **concevoir un premier modèle de segmentation d'images** qui devrait s'intégrer facilement dans la chaîne complète du système embarqué.

Du point de vue technique, l'objectif était de **fournir pour chaque image un « mask » exploitable** par le système de décision (succédant à l'étape de segmentation). Une fois le meilleur modèle décelé, nous étions amenés à le déployer en mettant en place une API. L'API, quant à elle, doit **prendre en entrée l'identifiant d'une image et renvoyer la segmentation** proposée par l'algorithme ainsi que celle de l'image réelle (son mask).

Segmentation sémantique

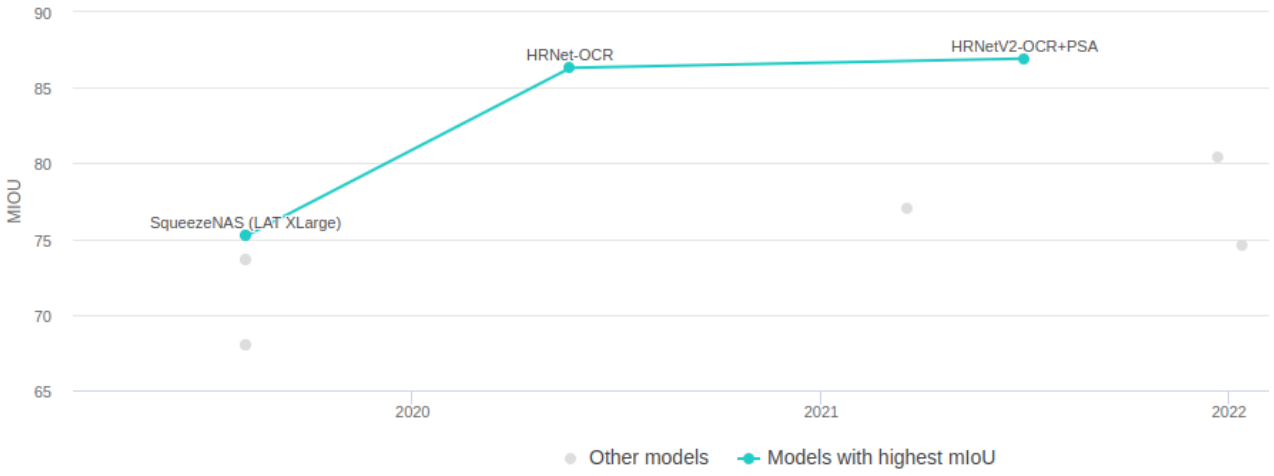
La segmentation d'images est issue du domaine scientifique **Computer vision**. Une segmentation consiste à **attribuer chaque pixel d'une image à un label.** Comment faire ? En divisant une image en plusieurs zones permettant la compréhension visuelle et l'analyse de l'image. Il existe plusieurs types de segmentations. Imaginons une image contenant plusieurs objets de catégories suivantes : humain, arbre, herbe ou ciel (4 labels).

- ➢ **la segmentation sémantique -** chaque pixel se voit attribuer un label. Ici, la segmentation détectera 4 labels, sans distinguer les différentes instances d'une même catégorie (tous les humains seront affichés avec la même couleur).
- ➢ **la segmentation d'instances -** c'est une combinaison de la segmentation sémantique et de détection d'objets. La détection de la cible vient en premier, puis chaque pixel est étiqueté (segmentation sémantique). Ici on distingue les différentes instances d'une même catégorie (les différentes personnes parmi les humains seront affichées chacun avec une couleur différente).
- ➢ **la segmentation panoramique** - consiste à détecter et à segmenter tous les objets de l'image, y compris l'arrière-plan, et à distinguer les différentes instances (différentes couleurs sont utilisées).

Or, notre besoin est de classer chaque pixel d'une image dans l'une des 8 catégories suivantes : *void, flat, sky, construction, object, nature, human, vehicule*. Puisqu'on se limite à l'attribution d'un label, sans distinguer ses instances, nous allons donc opter pour la segmentation sémantique.

2. Etat de l'art

Conformément à l'article « Semantic Segmentation on Cityscapes val [1]», l'état de l'art en matière de segmentation sémantique est obtenu grâce au modèle **HRNetV2-OCR+PSA,** qui **atteint une *mean IoU* de 86.93%** ! Le deuxième meilleur résultat est très proche : une *mean IoU* de 86,3% pour **HRNet-OCR.** En troisième position se place **ViT-Adapter-L** avec une *mean IoU* de 85.8%.



Evolution de l'état de l'art pour la segmentation sémantique sur le jeu de données Cityscapes.

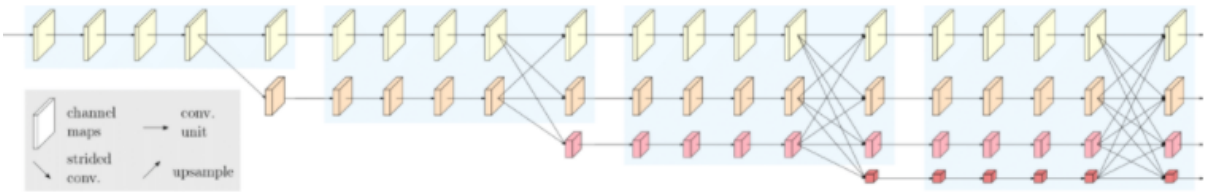
HRNet

Il s'agit d'architecture de type **Convolutional Neural Networks ou CNN**, développée en 2019 par Microsoft. Les CNN sont actuellement à la pointe de la technologie en vision par ordinateur et donc largement utilisés dans différentes tâches de reconnaissance d'images.

Très performant, le **HRNet** (*high-resolution network*) est une architecture universelle de reconnaissance visuelle. HRNet présente des performances supérieures ou compétitives sur un large éventail de **tâches sensibles à la position,** y compris la détection d'objets, l'estimation de la pose humaine, la détection de visages et la détection de repères faciaux.

Contrairement aux méthodes de type U-Net (downsampling puis upsampling), **le HRNet maintient des représentations de haute résolution tout au long du processus.** Par

conséquent, la représentation ainsi obtenue est plus riche sémantiquement et plus précise spatialement.



Architecture du HRNet configurée pour une segmentation sémantique

Il existe plusieurs versions de HRNet : la V1 s'applique à l'estimation de la pose humaine, la V2 à la segmentation sémantique et la V3 à la détection d'objets ou segmentation d'instances.

HRNet + OCR

OCR, *Object-Contextual Representations,* est *l'agrégation pondérée de toutes les représentations des régions d'objets avec les poids calculés selon les relations entre les pixels et les régions d'objet* [4]. Autrement dit, il s'agit de **caractériser un pixel en exploitant la représentation de la classe de l'objet** correspondante. C'est un algorithme que l'on applique après une segmentation sémantique pour en améliorer les performances.

HRNet + OCR + PSA

Le mécanisme PSA, quant à lui, combine deux blocs (connectés en parallèle) qui permettent au modèle **d'apprendre des informations sémantiques au niveau des pixels**, afin qu'il puisse obtenir la carte de profondeur avec des limites plus précises [5] :

- ➢ auto-attention au niveau des canaux (3 si RGB)
- ➢ auto-attention spatiale

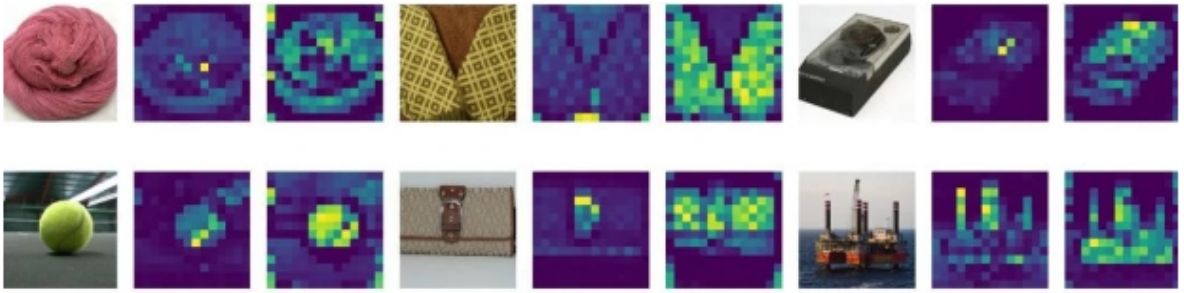
C'est donc cette architecture - HRNet V2 enrichi de mécanismes d'analyse du contexte (OCR) et d'informations sémantiques en profondeur (PSA) qui a remporté la première place de l'état de l'art 2022.

ViT

En 2022, **le Vision Transformer** (ViT) est apparu comme **une alternative compétitive aux réseaux de neurones convolutifs.** Les modèles ViT surpassent l'état de l'art actuel (CNN) de **près de x4 en termes d'efficacité et de précision de calcul.**
Le ViT est un modèle visuel basé sur **l'architecture d'un transformer** conçu à l'origine pour les tâches textuelles. Le modèle ViT représente une image d'entrée sous la forme d'une

série de patchs d'image et prédit directement les classes. ViT présente des performances extraordinaires lorsqu'il est formé sur une quantité de données suffisante.

CNN utilise des tableaux de pixels, tandis que ViT **partitionne les images en tokens visuels**. Le transformer visuel divise une image en patchs de taille fixe, intègre correctement chacun d'eux et inclut l'intégration positionnelle comme entrée de l'encodeur du transformer.



Exemples d'images brutes (à gauche) avec cartes d'attention du modèle ViT-S/16 [8]

3. Méthodologie

Dans les chapitres suivants nous allons comprendre le jeu de données Cityscapes, étudier les différentes fonctions de perte ainsi que certaines architectures de la bibliothèque **Segmentation models**.

Au départ, notre approche était la suivante : préparer les données, tester plusieurs architectures avec les différents backbones (avec les données augmentées et non augmentées). Malheureusement, suite à des problèmes techniques, un seul backbone a pu être testé - Resnet152.

3.1. Jeu de données

Le jeu de données vient de **Cityscapes**. Il contient de diverses scènes de paysages urbains enregistrées à différents moments de l'année et venant de 50 villes différentes.

Le jeu de données contient **25000 photos**, pourvues de masks de segmentations (autrement dit annotations). 5000 de ces masks sont finement annotés, tandis que les 20 000 masks restants sont beaucoup plus approximatifs. Pour notre projet, nous avons travaillé avec les photographies disposant d'un mask de catégories finement annoté.

Les paires **images - mask** sont pré-divisées en train (2975 images), validation (500 images) et test (1525 images) sets.

En termes de taille, les photographies et les masks qui leur sont associés sont tous proposés en résolution 2048 x 1024.

Prétraitement

Le jeu de données *Cityscapes* pose plusieurs problèmes.

Premièrement, **l'extraction de masques** - il a fallu extraire uniquement les fichiers labellisés *_gtFine_labelIds*. Nous avons opté pour une uniformisation de la nomenclature, pour qu'un masque ait le même nom que l'image d'origine.

Deuxièmement, la taille des images les rend **trop volumineuses.** Etant donné les moyens limités et de nombreux entraînements de modèles à effectuer, nous avons été contraints de **réduire la taille à 158 x 256.**

Ensuite, le jeu de données propose des annotations qui couvrent 8 catégories et 32 sous-catégories. Il s'agit d'objets couramment rencontrés lors d'une conduite. L'objectif était de **se focaliser sur les 8 catégories principales**. Pour cela, nous avons affecté chaque sous-catégorie à l'une des 8 classes-cibles, puis converti les masks :

- ➢ Vehicule
- ➢ Human
- ➢ Sky
- ➢ Nature
- ➢ Object (ex. panneaux de signalisation)
- ➢ Construction
- ➢ Flat (route)
- ➢ Void (autres ou problèmes de labellisation)

Finalement, nous avons procédé à une **normalisation d'images** (les valeurs RGB ont été divisées par 255.0).

3.2. Gestion du flux de données

Il est important de tenir compte de **l'instabilité du système d'acquisition d'images.** Il s'agit d'images enregistrées à différents moments de l'année dans un paysage urbain, nous avons donc affaire à un grand flux de données entrant qui, de plus, risque d'être biaisé suite à un défaut de prise d'image.

Pour remédier à ce problème, nous allons essayer de **reproduire ces conditions** en ayant recours d'une part à des techniques **d'augmentation des données,** et d'autre part à un **générateur de données**.

Data Generator

Etant donné la nature de l'input, il a fallu trouver un moyen pour **ne pas charger toutes les images** pour l'entraînement. La solution est **la classe Data Generator.** L'avantage des générateurs c'est qu'ils ne calculent pas la valeur de chaque élément. En effet, ils calculent les éléments uniquement lorsqu'on leur demande de le faire. Elle permet d'utiliser

immédiatement les données déjà calculées, pendant que le reste des données est en cours de calcul. Les générateurs permettent donc un gain de rapidité et d'espace mémoire.

La classe *Data Generator* a permis donc de **distribuer les images par batchs** (de taille paramétrable) et en parallèle appliquer les éventuels pré-processing ou augmentation de données.

Augmentation des données

Un autre aspect de la gestion du flux de données est leur instabilité. Mais comment reproduire un défaut de prise d'image ? Il faut pouvoir **réaliser des transformations sur les images** (ainsi que sur les masques de segmentation). L'idée est de **changer l'emplacement des pixels**, c'est-à-dire créer des images artificielles en ayant recours à la **Data Augmentation.**

L'idée derrière la *Data Augmentation* est de reproduire les données préexistantes en leur **appliquant une transformation aléatoire.** Plusieurs avantages suite à cela :

- ➢ augmentation de la taille du jeu de données
- ➢ réduction de risques de sur-apprentissage

Lors de l'entraînement, notre modèle apprendra sur beaucoup plus de données tout en ne rencontrant jamais deux fois la même image.

Pour cela nous avons fait appel à **la librairie IMGAUG** qui permet de nombreuses transformations. Ci-dessous quelques exemples :

Exemple d'image transformée : photo d'origine, Blur, Zoom, Brightness.

3.3. Métriques

Pour pouvoir évaluer l'efficacité d'un modèle de segmentation sémantique, il faut être capable de dire quand est-ce qu'une prédiction est "vraie" et quand est-ce qu'une prédiction est "fausse".

La mesure classique - **Accuracy** - est à proscrire, car les résultats peuvent être « trop » bons en cas de déséquilibre de classes. Si par exemple la classe « background » couvre 90% de l'image, on obtient une précision de 90% en classifiant tous les pixels comme «background ».

En effet, l'idée serait de **calculer le recouvrement** qui existe entre le résultat prédit et le résultat attendu :

Calcul du recouvrement : en rouge, la prédiction - en vert, la vérité terrain.

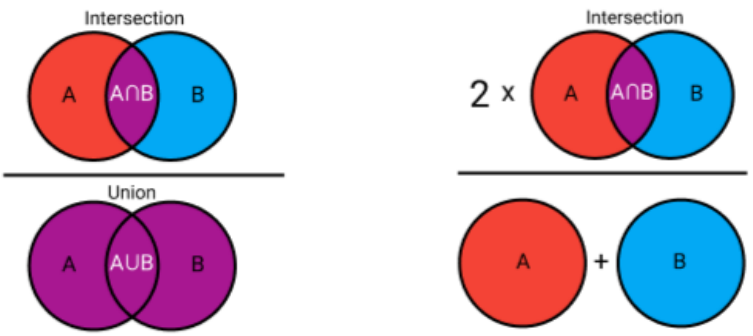
Il existe deux mesures permettant ce calcul :

➢ **Le coefficient de Dice** (ou indice de Dice-Sørensen, ou score F1)

Le coefficient se définit comme le double de l'intersection de deux lots (échantillons de valeurs) divisé par l'union de ces deux lots (cf la figure plus bas).

➢ **Le coefficient de Jaccard** (IoU ou Intersection Over Union)

L'IoU représente le ratio entre l'intersection du masque de segmentation réel et du masque prédit (vrais positifs), et l'union des 2 masques (vrais positifs + faux positifs + faux négatifs). Dans le cas de classes multiples, l'IoU de chaque classe est calculé et on prend leur moyenne.



Les formules des métriques : IoU versus Dice-Sorensen.

Pour les deux mesures, on obtient un score entre 0 (pas d'intersection du tout) et 1 (masques identiques).

La métrique que nous avons surveillée de près est l'IoU. C'est la métrique de référence utilisée pour comparer les modèles de l'état de l'art. Nous avons noté également le score F1 (Dice) tout au long de nos tests.

3.4. Fonctions de perte

La segmentation d'image peut être considérée comme une tâche de classification au niveau du pixel. **Le choix de la fonction de perte pour cette tâche est essentiel** pour déterminer à la fois la vitesse à laquelle un modèle d'apprentissage automatique converge, ainsi que, dans une certaine mesure, la précision du modèle. Elle est utilisée pour **guider un modèle dans sa recherche de l'approximation "idéale"** qui fait correspondre les données d'entrée aux données de sortie (des images aux masques dans notre cas). [9]

Notre méthodologie présume l'entraînement du modèle de référence (baseline) avec à chaque fois une modification de l'hyper paramètre *loss*.

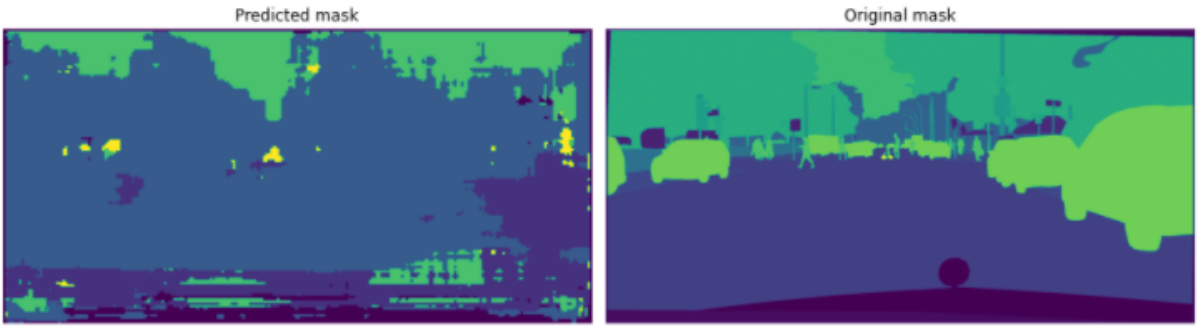
Baseline

C'est une architecture simple **de type UNET** de la librairie Keras qui nous a servi de modèle de référence.

Les fonctions de perte testées sont les suivantes :

- **Dice Loss** - utilisée pour calculer la similarité entre les images (à l'image de l'approche d'Intersection-over-Union). Sa version pondérée (**Weighted Dice Loss**) permet d'attribuer plus de poids aux classes minoritaires (comme les piétons ou véhicules).
- **Focal Loss** - permet de pallier le problème de déséquilibre des classes : réduit la perte pour les exemples bien classés (proba > 0.5), augmente la perte pour les exemples difficilement classifiables (proba < 0.5)
- **BCE Jaccard Loss (IoU) -** la « binary cross entropy » combinée à la fonction Jaccard (inspirée du coefficient de Jaccard), utilisée pour des datasets déséquilibrés
- **Categorical Focal + Dice Loss** - une combinaison des categorical_focal_loss et dice_loss qui est censée être très robuste

Au bout de plusieurs essais, nous sommes parvenus à identifier la fonction de perte la plus adaptée à notre problème : la **Categorical Focal + Dice Loss** est celle qui s'en est le mieux sortie pour prédire le mask de l'image-test :



Ci-dessous le tableau avec les résultats :

	IoU	Dice	training_time
Base-Categ-Focal-Dice	0.496615	0.601135	7261.540968
Base-Dice	0.513233	0.622157	7420.738188
Base-Weight-Dice	0.387483	0.507299	7506.385689
Base-Focal	0.423833	0.539918	7538.611242
Base-Jaccard	0.449309	0.561707	7876.374853

C'est donc avec *categorical_focal_dice_loss* que nous avons entraîné les modèles suivants.

4. Modélisations

Une fois la loss function la plus performante identifiée, il est temps d'évaluer les modèles de segmentation sémantique.

Classification d'images versus Segmentation sémantique

Par rapport aux tâches de classification ou de détection d'objets, la segmentation est une tâche beaucoup plus difficile.

Dans **la classification**, il s'agit de **classer l'objet au sein d'une image** : une image d'entrée est réduite et passe par les couches de convolution puis les couches entièrement connectées (FC), pour **produire en sortie une étiquette prédite** pour l'image d'entrée [11].

L'objectif est de cartographier le tensor spatial (obtenu par les couches de convolution) par un vecteur de longueur fixe. La présence des couches FC provoque une destruction de toute information spatiale.

Dans **la segmentation sémantique**, il s 'agit **d'attribuer une classe** à **chaque pixel d'une image**. Cela signifie qu'il y a **une étiquette pour chaque pixel.**

Ici, il est primordial de préserver l'information spatiale, donc aucune couche entièrement connectée n'est utilisée (il s'agit de réseaux entièrement convolutionnels).

Architectures de segmentation d'images

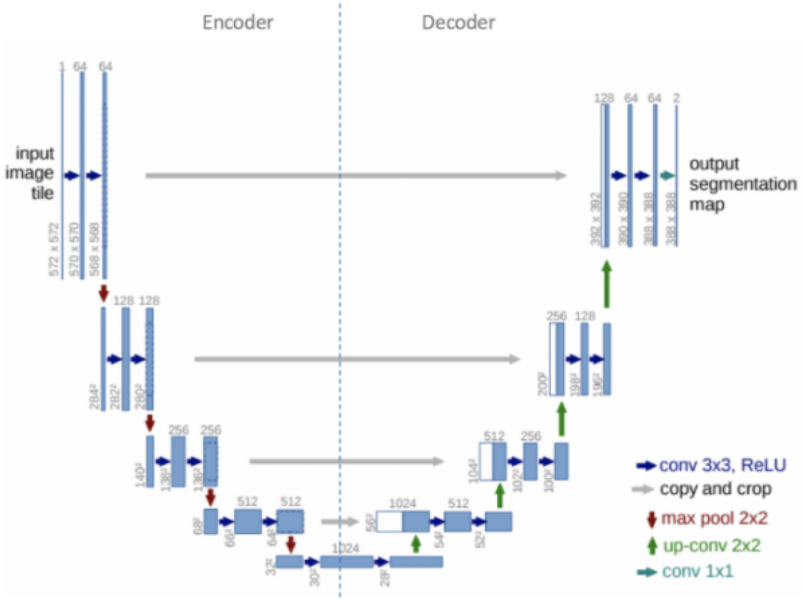
Il existe un grand nombre d'architectures conçues pour ce genre de problématique. Pour nos besoins nous avons fait appel aux modèles basés sur le framework Keras (Tensorflow) de la bibliothèque Python **Segmentation models**, qui propose plusieurs architectures : **U-Net, LinkNet, PSPNet et FPN**.

Segmentation models

La majorité des architectures de segmentation d'images utilisent la **stratégie d'encodage-décodage**. L'encodeur **extrait les caractéristiques de l'image** (features) à travers des filtres et le décodeur mappe ces informations dans une catégorisation spatiale pour **générer la sortie finale** (généralement un masque de segmentation). Lors de la phase de l'encodage, l'image est fortement sous-échantillonnée. Le problème majeur de la segmentation sémantique est de suréchantillonner la carte de features à la résolution d'origine et de préserver la catégorisation des pixels.

U-NET

L'encodeur - la partie gauche - est basé généralement sur un réseau convolutionnel comme Resnet (backbone). Il utilise des **blocs de convolution suivis d'un downsampling** pour créer une carte de caractéristiques d'une image et de réduire sa taille pour diminuer le nombre de paramètres du réseau.



Le schéma d'architecture U-Net [13].

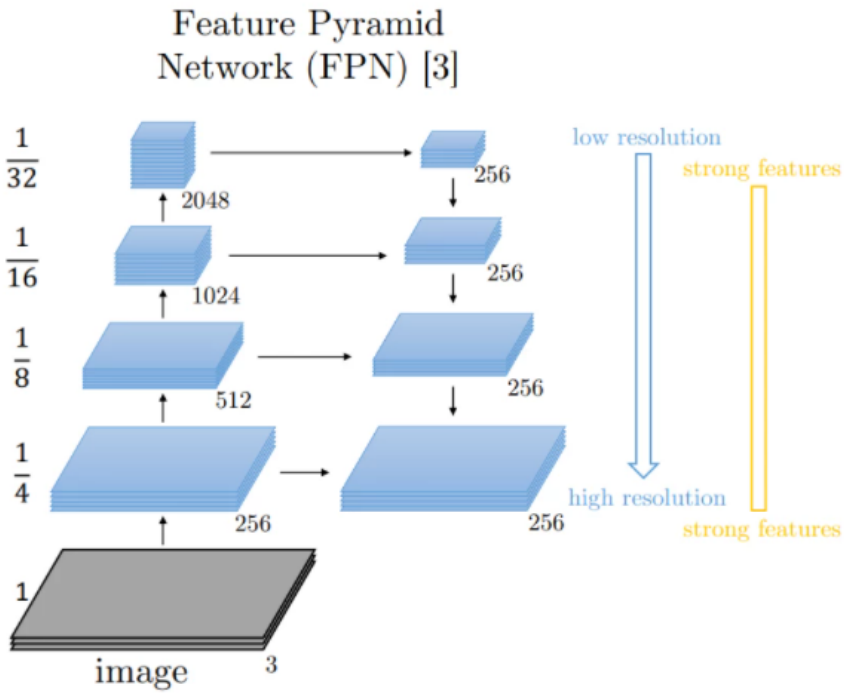
Le décodeur, quant à lui, a deux particularités : premièrement, **il permet un upsampling** de la carte de caractéristiques au point de récupérer les dimensions d'entrée (la *déconvolution)*. Deuxièmement, il permet **une localisation précise de features** grâce à la convolution transposée (*skip connections*). L'objectif étant d'éviter des pertes d'information, on laisse le décodeur avoir accès aux features de bas niveau produites par les couches de l'encodeur (grâce aux **skip connections**).

Ensuite une concaténation a lieu : les résultats de l'encodeur sont ajoutés aux entrées des couches du décodeur, à des endroits appropriés. Finalement, une convolution 1x1 est utilisée dans la couche finale pour **attribuer les canaux au nombre requis de classes**.

FPN (Feature Pyramid Networks)

Dans FPN, la partie encodage ressemble à celle de U-Net. Pour la partie décodage, FPN produit une image de sortie pour chaque **étage de la pyramide**, d'où son nom. Concrètement, **on combine les cartes de features** à basse résolution qui ont des caractéristiques sémantiquement fortes, avec l'image précédemment sur-dimensionnée qui a des caractéristiques sémantiquement faibles.

Nous obtenons des **caractéristiques plus fortes à mesure que nous progressons** dans le réseau neuronal. Par exemple, dans la première couche du réseau CNN, nous pouvons trouver des caractéristiques telles que des lignes ou de simples bords d'un objet, mais à mesure que nous approfondissons, nous pouvons trouver des caractéristiques décrivant l'image, comme un piéton ou une voiture.

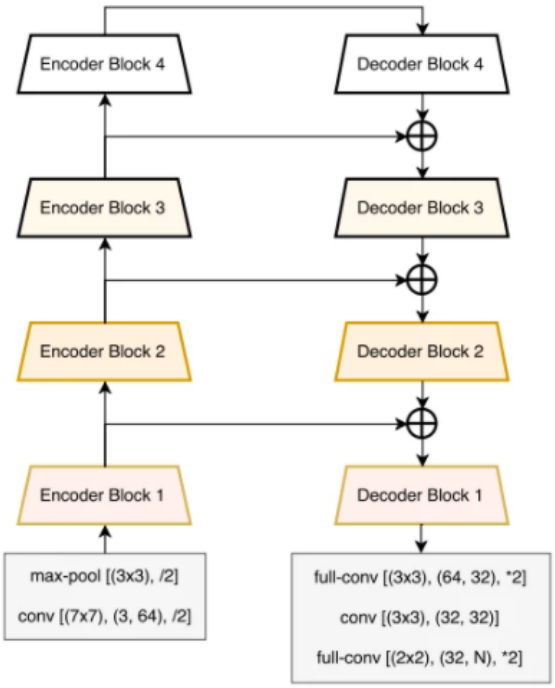


Le schéma d'architecture FPN [13].

Pour cette raison, la **pyramide des features** possède de riches fonctionnalités sémantiques à tous les niveaux [13].

Linknet

Dans LinkNet, l'entrée de chaque **couche d'encodeur est également transmise à la sortie de son décodeur** correspondant. Grâce à cela, les informations spatiales perdues sont récupérées et peuvent être utilisées par le décodeur et ses opérations d'upsampling.



Le schéma d'architecture Linknet [13].

La figure ci-dessus permet de voir le passage de l'information encodée au bloc décodeur correspondant.

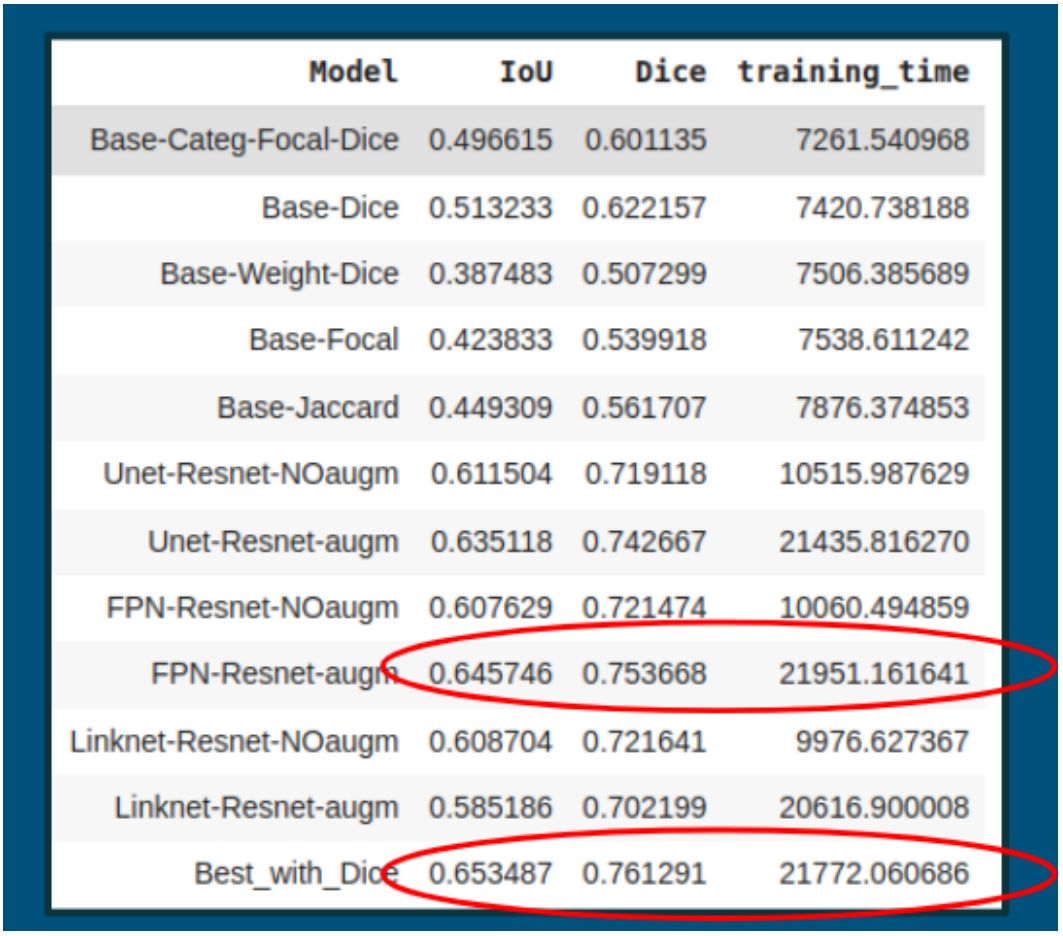
5. Synthèse des résultats

Il est temps d'afficher le tableau récapitulatif ! Quel modèle a obtenu le score IoU (et Dice) le plus élevé ? (cf plus bas *Tableau récapitulatif*)

Tous les modèles ont reçu en entrée un échantillon de 70% de données initiales.

Le modèle le plus rapide à l'exécution est `**Linknet**` sans augmentation de données.
Le modèle qui a tourné le plus longtemps est le `**FPN**` avec augmentation de données.

Il est intéressant de constater que, de manière générale, les données augmentées ont de meilleurs résultats, sauf pour l'architecture `**Linknet**`. On peut en conclure que même si les concepts des trois architectures sont similaires, les résultats sont assez différents.



Model	IoU	Dice	training_time
Base-Categ-Focal-Dice	0.496615	0.601135	7261.540968
Base-Dice	0.513233	0.622157	7420.738188
Base-Weight-Dice	0.387483	0.507299	7506.385689
Base-Focal	0.423833	0.539918	7538.611242
Base-Jaccard	0.449309	0.561707	7876.374853
Unet-Resnet-NOaugm	0.611504	0.719118	10515.987629
Unet-Resnet-augm	0.635118	0.742667	21435.816270
FPN-Resnet-NOaugm	0.607629	0.721474	10060.494859
FPN-Resnet-augm	0.645746	0.753668	21951.161641
Linknet-Resnet-NOaugm	0.608704	0.721641	9976.627367
Linknet-Resnet-augm	0.585186	0.702199	20616.900008
Best_with_Dice	0.653487	0.761291	21772.060686

Tableau récapitulatif avec les résultats de modélisations

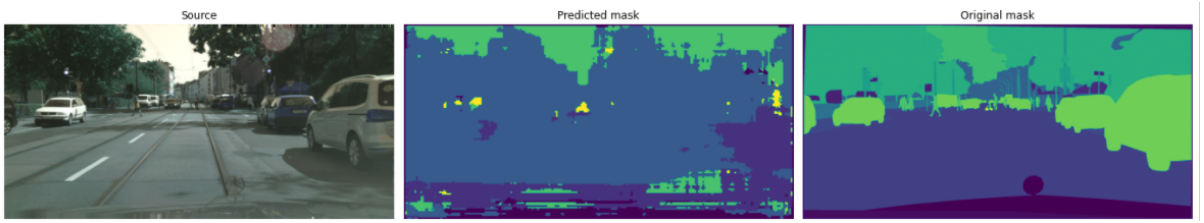
Le modèle qui mérite une attention particulière est le `**FPN-Resnet-augm**` avec la fonction de perte `Categorical Focal + Dice` qui obtient un bon score (IoU de 0.64, Dice de 0.75, temps d'exécution : 6h). Le même modèle entraîné avec la fonction de perte `Dice` (`Best_with_Dice`) obtient les scores légèrement supérieurs : IoU de 0.65, Dice de 0.76 et les temps d'exécution également de 6h.

Au final, c'est donc une **architecture FPN basée sur un backbone Resnet152 et la fonction de perte Dice Loss** qui a été retenue pour le déploiement.

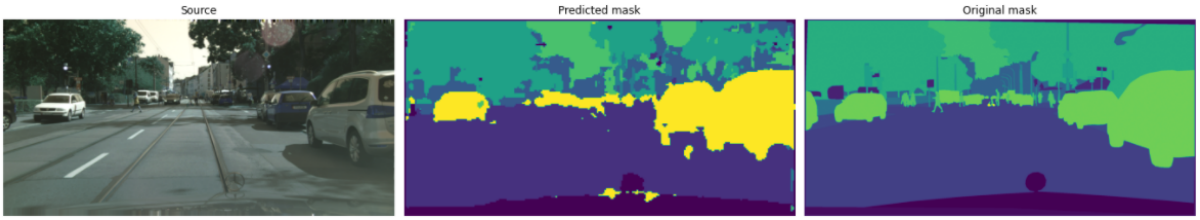
Simulation de prédiction

Ci-dessous un exemple de prédiction pour le modèle choisi et la baseline :

Baseline



Modèle final



6. Conclusions

Les prédictions obtenues par notre modèle final laissent une marge d'amélioration. Il existe plusieurs pistes d'amélioration envisageables pour booster les scores.

Données

En termes de données, l'input pourrait être **plus volumineux**. Etant donné les ressources limitées, nous avons utilisé un sample de 50% pour tester les fonctions de perte et 70% pour tester les différentes architectures.

Un changement de **résolution d'images** aurait pu être également effectué : 512 x 1024 au lieu de la taille réduite 128 x 256.

Nous constatons que le fait de procéder à **une augmentation de données** donne de meilleurs résultats. Une modification de techniques d'augmentation pourrait rendre les résultats plus pertinents.

Modèles

D'autres architectures et surtout **d'autres backbones** sont à tester absolument : *VGG16 ou VGG19, Densnet, Inception, ResNext, EfficientNet* etc.
Il existe également d'autres **fonctions de perte** qui n'ont pas été testées dans le cadre de ce projet et qui auraient pu améliorer le modèle.

Références

[1] https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes-val
[2]https://www.microsoft.com/en-us/research/blog/high-resolution-network-a-universal-neural-architecture-for-visual-recognition/
[3]https://medium.com/@ltylty221/object-contextual-representations-for-semantic-segmentation-abe78e422fac
[4] https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123510171.pdf
[5] https://www.sciencedirect.com/science/article/abs/pii/S0925231222009109
[6] https://machinelearningmastery.com/the-vision-transformer-model/
[7] https://viso.ai/deep-learning/vision-transformer-vit/
[8] https://arxiv.org/abs/2106.01548
[9] https://dev.to/_aadidev/3-common-loss-functions-for-image-segmentation-545o
[10] https://makina-corpus.com/data-science/extraction-objets-cartographie-deep-learning
[11] https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1
[12] https://github.com/qubvel/segmentation_models
[13] https://hasty.ai/docs/mp-wiki/model-architectures/