

Note Technique



Segmentation d'images pour la conception de voitures autonomes



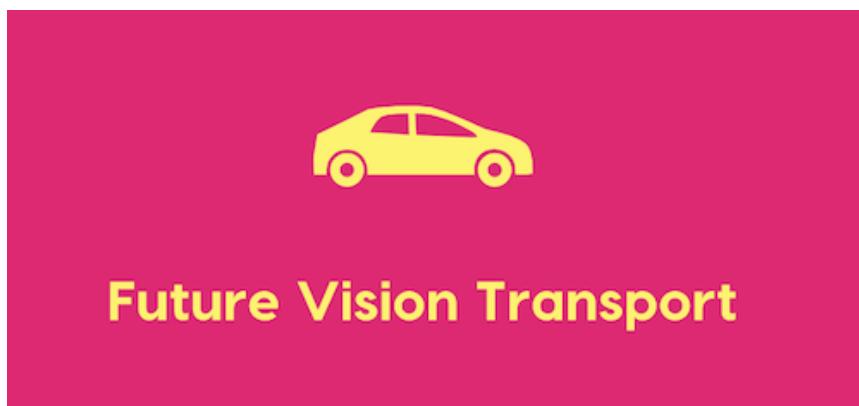
Nicolas Blanchon

I. Introduction

A. Présentation du projet

Dans le cadre de la stratégie de développement de la société Future Vision Transport, qui conçoit des systèmes embarqués de vision par ordinateur pour les véhicules autonomes. L'équipe R&D est missionnée sur le développement et l'industrialisation du système, selon 4 parties distinctes:

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



Au sein de la section “Segmentation des Images”, nous avons en charge la conception d'un modèle de segmentation d'images, qui servira ensuite de point d'entrée pour le système de décision.

B. Objectifs

L'objectif est de fournir une API qui prend en entrée une image, et renvoie la segmentation sous forme de masque.

Pour se faire, nous travaillerons sur 3 axes distincts:

- Entrainement de modèles de segmentation d'images:
 - 8 Catégories identifiables
 - Reposant sur la librairie Keras de Tensorflow
- Conception d'une API de prédiction
 - Accessible via le Cloud
 - Exposition du modèle Keras
- Conception d'une Interface Web
 - Accessible via le Cloud
 - Simplicité d'utilisation avec 4 fonctionnalités majeures

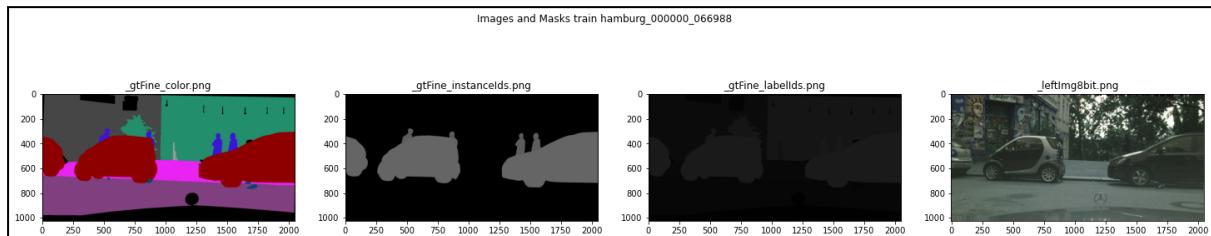
C. Analyse des données

Les données Cityscapes sont à télécharger via le site suivant:
<https://www.cityscapes-dataset.com/dataset-overview/>

Une fois les 12Go stockées, Les images de tailles 2048*1024 sont architecturées dans une arborescence de dossier comme suit:

- `gtFine/` : Dossier contenant les images transformées
 - `test/`, `train/` and `val/` : 3 Dossiers pour différencier les jeux de données
 - `berlin/`, `frankfurt/`, `munich/`, ... : Dossiers par villes, contenant les images
 - `*_gtFine_color.png` : Images colorées en RGB
 - `*_gtFine_instanceIds.png` : Images en variations de gris, suivant les InstanceIds
 - `*_gtFine_labelIds.png` : Images en variations de gris, suivant les labelIds de -1 à 32.
- `leftImg8bit` : Dossier contenant les images natives
 - `test/`, `train/` and `val/` : 3 Dossiers pour différencier les jeux de données
 - `berlin/`, `frankfurt/`, `munich/`, ... : Dossiers par villes, contenant les images
 - `*_leftImg8bit.png` : Images natives variations de gris

Voici ci-dessous une représentation des images associées:



Les 8 Catégories recherchées pour notre segmentation sont décrites sur le [site web de CityScapes](#):

1. void
2. flat
3. construction
4. object
5. nature
6. sky
7. human
8. vehicle

A des fins d'entraînement du modèle, nous utiliserons les données "Train" provenant de:

- *leftImg8bit* en tant que données d'entrées : Image à analyser
- *_gtFine_labelIds* regroupé sur 8 classes¹, en tant que données Cibles : Masque

¹ Mapping permettant de passer des 32 sous-catégories aux 8 catégories

II. Vision par ordinateur

La vision par ordinateur est un domaine scientifique et branche de l'intelligence artificielle qui traite de la façon dont les ordinateurs peuvent acquérir une compréhension de haut niveau à partir d'images ou de vidéos numériques.

Les techniques de Computer Vision reposent sur l'entraînement de modèles supervisés. Cela consiste à entraîner un réseau de neurones sur des couples (X=Images, Y=Masques).

A. Classification

En vision par ordinateur on désigne par classification d'objet une méthode permettant de détecter la présence d'une instance (reconnaissance d'objet) dans une image numérique.

CLASSIFICATION



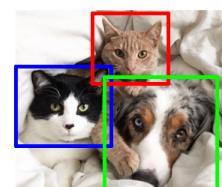
CHAT

La tâche la plus courante de classification d'image résulte à identifier une classe au sein de l'image (Ici, l'image représente un chat).

B. Détection d'objet

La Détection d'objets consiste à identifier tous les objets présents sur l'image, ainsi que leur position et leur type. Une image peut ainsi comporter plusieurs catégories.

DÉTECTION D'OBJET



CHAT, CHAT, CHIEN

MULTIPLE OBJET

La détection est souvent définie par des rectangles, appelés bounding box.

C. Segmentation sémantique

La Segmentation consiste à classer chaque pixel d'une image selon leur catégorie.



CHAT, DRAP, CHIEN

On ne peut faire de différence entre deux instances d'un même objet.
Par exemple, si on a deux voitures sur une image, ce type de segmentation donnera le même label sur l'ensemble des pixels des deux voitures.

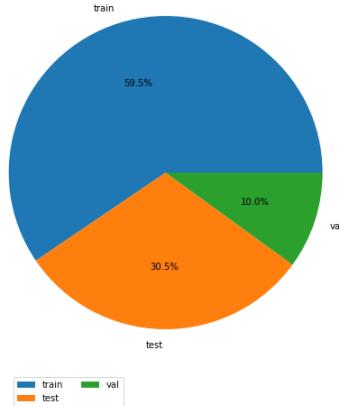
C'est cette segmentation sémantique que nous utiliserons lors du projet.

III. Modélisation

A. Génération des données

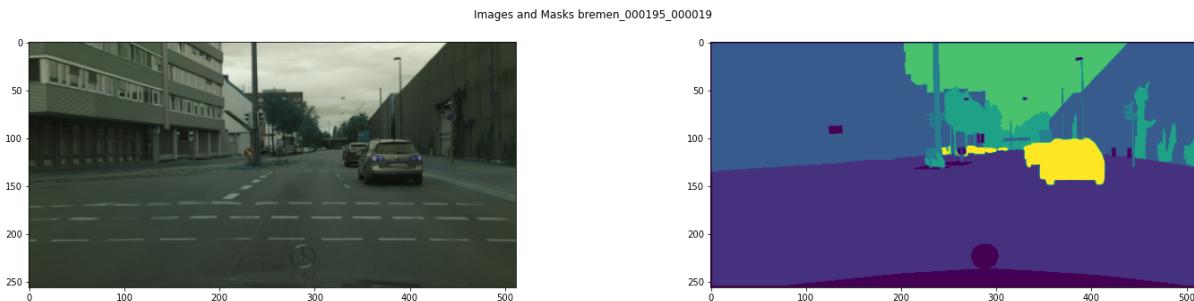
Au regard de la taille importante du volume de données, il est préférable d'utiliser un générateur pour charger l'ensemble de ses données à son réseau de neurones via des mini-lots de données (Batch). Ceci évite d'envoyer le dataset dans sa globalité et ainsi saturer la mémoire allouée.

3 most presents values identified in column Phase .
TOTAL unique = 3



Phase	
train	2975
test	1525
val	500

Le générateur de données retourne les images et les masques associés pour chaque batch.



Il est constitué d'un squelette sous forme de “classe” python customisée héritant de Keras.utils.Sequence et permettant le multiprocessing (calcul parallèle avec nos threads).

Nous retrouverons les étapes de conception du générateur dans le fichier notebook: *data_generator.ipynb* et *Functions_custo_by_Nico.ipynb*.

1. Création d'un dataframe

En préambule de la génération de donnée, nous avons construit un dataframe pandas listant chacune des images présentes dans les dossiers/sous-dossiers, ainsi que leur emplacement et leur caractéristiques (train/val/test, unique id, city).

Ce dataframe nous servira de point d'entrée pour générer des batch alimentant notre modèle de réseau de neurone lors de son entraînement.

Nous l'avons sauvegardé en tant que fichier csv: *clean_df.csv*.

2. Lecture des Images

Dans le cadre de segmentation d'images, nous avons choisi d'utiliser la librairie [OpenCV](#) pour lire, traiter et modifier les images (exemple: tailles des images).

Nous évaluerons ci-après les effets de modifications des images par le générateur.

Note: Afin de minimiser les ressources employées lors de notre phase de test, nous avons réduit nos images comme suit: Hauteur: 128 pixels * Largeur: 256 pixels.

3. Label Encoding

Notre objectif est d'identifier les 8 classes distinctes décrites précédemment. Pour se faire, nous avons la possibilité d'affecter une valeur entre 0 et 7 pour chaque pixel.

4. OneHotEncoding

One hot encoding consiste à définir un ensemble de colonnes selon le nombre de classes possibles. Chacune des 8 colonnes représente une seule classe. Seule la colonne représentant la classe aura comme valeur 1. Les autres colonnes auront comme valeur 0.

Dans la suite du projet, nous avons utilisé le OneHotEncoding (besoin associé à la fonction de coût categorical_crossentropy décrite ci-dessous), suivi de la fonction argMax de Numpy pour récupérer la classe prédite.

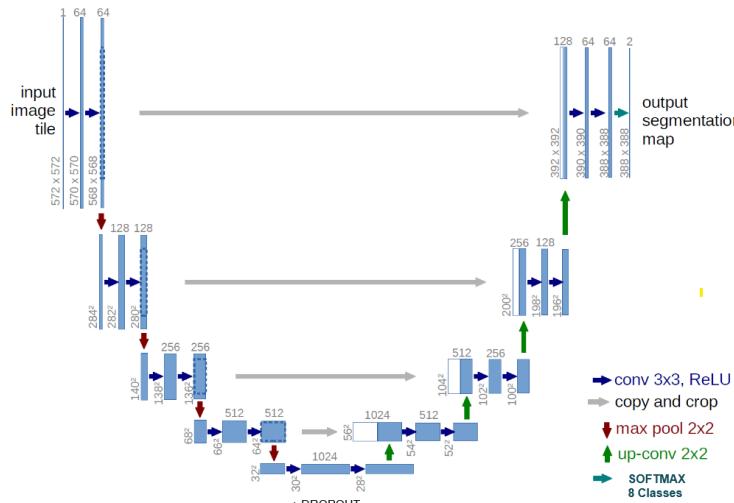
B. Modèles de réseaux de neurones

1. UNET Baseline

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

L'architecture de U-NET est composée de deux chemins (architecture en U):

- Contraction: Capturer le contexte d'une image.
- Expansion: Localise grâce à la convolution transposée.



Cette architecture préserve la taille initiale = la taille de sortie est égale à la taille d'entrée.

Nous concevons ici notre propre modèle UNET, en utilisant les APIs [Keras](#) de la librairie Tensorflow.

Nous adaptons l'architecture avec une fonction d'activation SOFTMAX pour de la prédiction multi classes (8).

2. UNET Backbones

Afin de comparer différents modèles, nous étudions également des modèles Backbone pré-entraînés, en se basant sur la librairie [segmentation-models](#) qui s'appuie également sur Keras.

Ces modèles ont pour avantages de contenir des poids pré entraînés sur les données Imagenet pour l'initialisation.

Ainsi, nous évaluerons les résultats que produisent un entraînement des modèles Backbones UNET avec EfficientNetB2 et ResNet34 sur nos données.

3. PSPNet et FPN

Toujours dans l'objectif d'optimisation du modèle, nous recherchons des améliorations au travers de nouvelles architectures ([PSPNet](#) et [FPN](#)). Ces nouvelles architectures apportent des modifications sur les connexions entre couches.

C. Comparaisons de résultats

1. Fonction de perte (coût)

Une fonction de perte, ou Loss function, est une fonction qui évalue l'écart entre les prédictions réalisées par le réseau de neurones et les valeurs réelles des observations utilisées pendant l'apprentissage. Plus le résultat de cette fonction est minimisé, plus le réseau de neurones est performant.

Afin d'optimiser l'apprentissage de notre modèle, nous avons évalué différentes fonctions de pertes:

- “categorical_crossentropy”: cette fonction vient directement de Tensorflow, elle permet de calculer l'entropie croisée de notre modèle multi-classes. Elle évalue une moyenne des résultats sur l'ensemble des pixels. De ce fait, cette fonction gère assez mal les datasets déséquilibrés, et ne correspond pas à notre problématique.
- “dice_loss”: cette fonction n'est autre que « 1 - le coefficient de dice » (Voir métrique DICE ci-dessous)
- “categorical_focal_loss”: Cette fonction réduit le poids des exemples faciles afin que le réseau se focalise plutôt sur les cas complexes.
- “combinaison 0.5*crossentropy + dice_loss”: facilite l'interprétation et considère le déséquilibre des classes.
- “combinaison 0.5*categorical_focal_loss + dice_loss”: considère la segmentation de cas complexe et le déséquilibre des classes.

2. Choix de la métrique

Il est primordial de choisir la métrique la plus adaptée au besoin métier, dans notre cas la segmentation sémantique d'images. A ce titre, nous avons évalué les métriques essentielles pour la mesure de la performance:

- “Accuracy”: Dans un premier temps, nous avons choisi la métrique la plus simple à comprendre, puisqu'elle mesure le pourcentage de pixel correctement classifié. Elle

ne s'avère en revanche pas adaptée à notre besoin, puisque nos images sont déséquilibrées (ex: faible proportion d'humain)

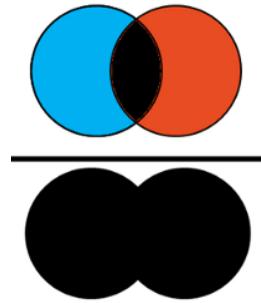
- “Mean IOU”: Intersection-Over-Union se base sur la matrice de confusion, et peut être représentée de la façon suivante :

$$\frac{TP}{(TP + FP + FN)}$$

Comme représenté, il s'agit de l'aire de chevauchement, sur l'aire d'union. Dans le cas d'une segmentation multiclasses à 8 catégories, il s'agit de la moyenne de chacune des 8 IOU.

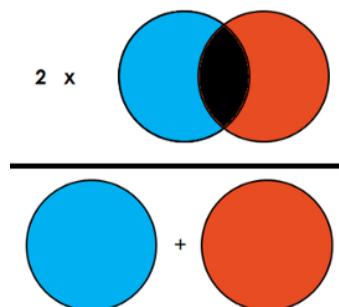
Le score est compris entre 0 et 1, 1 étant la similarité maximum entre la prédiction et la réalité de la segmentation.

Note: Il est aussi appelé “Jaccard Index”



- “Coefficient de DICE”: Il est assez similaire à l'IOU, mais se montre un peu moins stricte sur les vrais négatifs (favorise les vrais positifs). Il peut être représenté de la façon suivante:

$$\frac{2TP}{(2TP + FP + FN)}$$



Dans le cas de l'entraînement de notre modèle, nous utiliserons le “Mean IOU”, mais nous mesurerons également le DICE.

3. Augmentation des données

Afin de maximiser nos chances de réussite, nous avons évalué l'apport de l'augmentation du dataset. Dans le cas des images, nous avons utilisé la librairie [Albumentations](#).

Ici, l'augmentation à pour but d'appliquer des transformations aux images d'origines, et permet donc l'obtention de nouvelles images pour l'apprentissage de notre réseau de neurones

Nous avons essayé les augmentations d'images suivantes:

- Rotation de l'image à 25°
- Random RGBShift (modification aléatoire des valeurs de canaux RGB)
- Random HueSaturationValue (modification aléatoire de la saturation)
- Random Clae (modification aléatoire du contraste de l'histogramme)
- Random Clae (modification aléatoire de la luminosité et du contraste)
- Random Gamma (modification aléatoire du gamma)

4. Meilleur modèle retenu

Suite au différents modèles étudiés et testés, nous pouvons identifier le meilleur modèle pour notre segmentation:

- Architecture: Backbone FPN avec poids pré-entraîné “efficientnet”
- Fonction de perte: combinaison $0.5 * \text{categorical_focal_loss} + \text{dice_loss}$
- Augmentation: Non activée
- Tailles des Images : Largeur 512 pixels * Hauteur 256 pixels

Ce modèle obtient un IOU score = 0.71347 ainsi qu'un résultat de fonction de perte = 0.2099.



Voici la liste des 7 meilleurs modèles:

Model	Fonction de perte	Largeur	Hauteur	Augmentation	Perte	Dice	IOU
FPN Backbone	CATFOCA L-DICE	512w	256h	No	0,210	0,801	0,713
FPN Backbone	CATFOCA L-DICE	512w	256h	Yes	0,211	0,799	0,710
UNET Backbone	CATFOCA L-DICE	512w	256h	No	0,216	0,795	0,707
FPN Backbone	CATFOCA L-DICE	256w	128h	Yes	0,245	0,768	0,667
FPN Backbone	CATFOCA L-DICE	256w	128h	No	0,256	0,760	0,659
FPN Backbone	CrossEntropy-DICE	256w	128h	Yes	0,282	0,761	0,657
UNET	CrossEntropy-DICE	256w	128h	No	0,276	0,758	0,652

IV. Prototypage

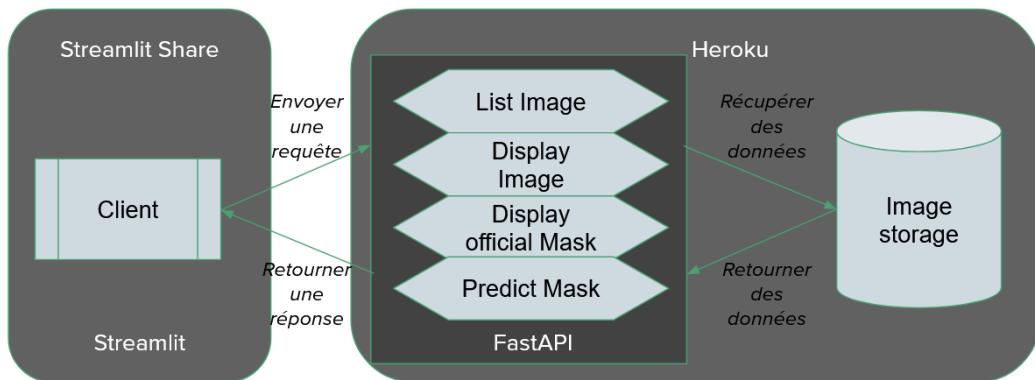
Afin de démontrer le fonctionnement de ce modèle, nous avons exposé nos ressources (modèles + Set d'images) sur le Cloud.

Nous avons ainsi déployé:

- 4 APIs reposant sur le framework FastAPI
- 1 Interface utilisateur reposant sur le framework Streamlit

A. Architecture Client / Serveur

Voici ci-dessous une représentation de notre architecture technique permettant de faire fonctionner notre prototype.



B. Déploiement en Production

L'URL pour accéder à la page Streamlit est:

<https://blanchonnicolas-ia-project8-openclassroom-user-interface-5iw4te.streamlitapp.com/>

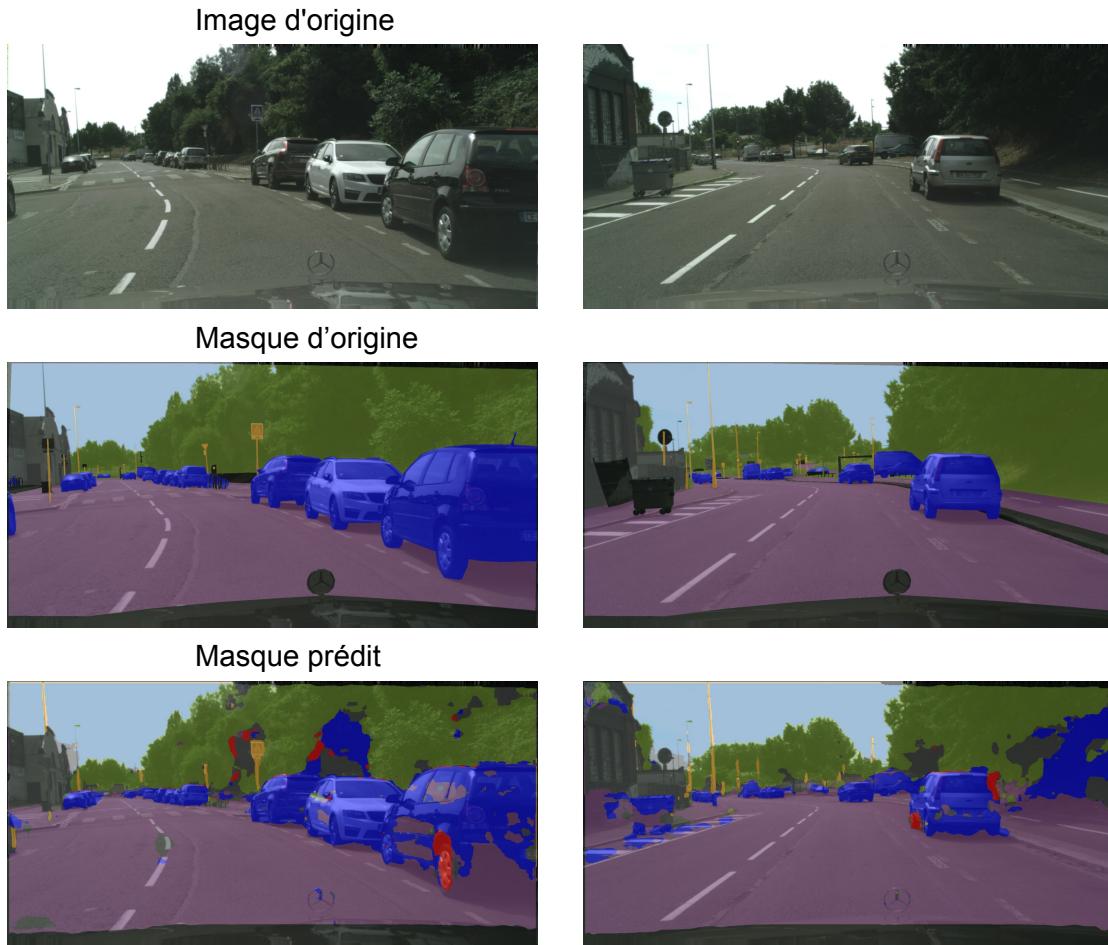
L'URL pour accéder à la documentation et aux fonctionnalités de l'API via Swagger sont:

<https://vast-sea-04286.herokuapp.com/docs#/>

Note: Le modèle déployé en production n'est pas le meilleur modèle obtenu à ce jour, afin de limiter les coûts d'infrastructure (Taille du modèle trop importante pour être sauvegardé sur une plateforme gratuite - Heroku).

V. Conclusions

Vous pouvez visualiser ci-dessous un extrait des images, masques et résultats de prédiction que vous pourrez obtenir en production.



VI. Pistes d'améliorations

Si les résultats et l'approche vous semble satisfaisante, nous pouvons poursuivre l'amélioration de nos modèles de la façon suivante:

- Chercher un apprentissage basé sur d'autres tailles d'images (plus grandes, donc nécessitant plus de ressources de calculs)
- Tester et sélectionner différentes augmentations des données
- Evaluer l'apport de combinaisons plus poussées pour les fonctions de pertes
- Mesurer les différents apports des profondeurs EfficientNet (B0 à B7)
- Augmenter la taille du jeu d'apprentissage.

VII. Sources & References:

https://fr.wikipedia.org/wiki/Vision_par_ordinateur
<https://deeplearning.fr/cours-theoriques-deep-learning/differences-entre-classification-dimage-localisation-detection-segmentation-dobjets/>
<https://deeplearning.fr/cours-pratiques-deep-learning/segmentation-semantique-dimages/>
<https://datascientest.com/u-net>
<https://lars76.github.io/2018/09/27/loss-functions-for-segmentation.html>
<https://deeplearning.fr/cours-theoriques-deep-learning/fonctions-de-perte-cout-loss-function/>
<https://www.editions-eni.fr/open/mediabook.aspx?idR=f6e7a7353a3574180124387fa03fdc1c>